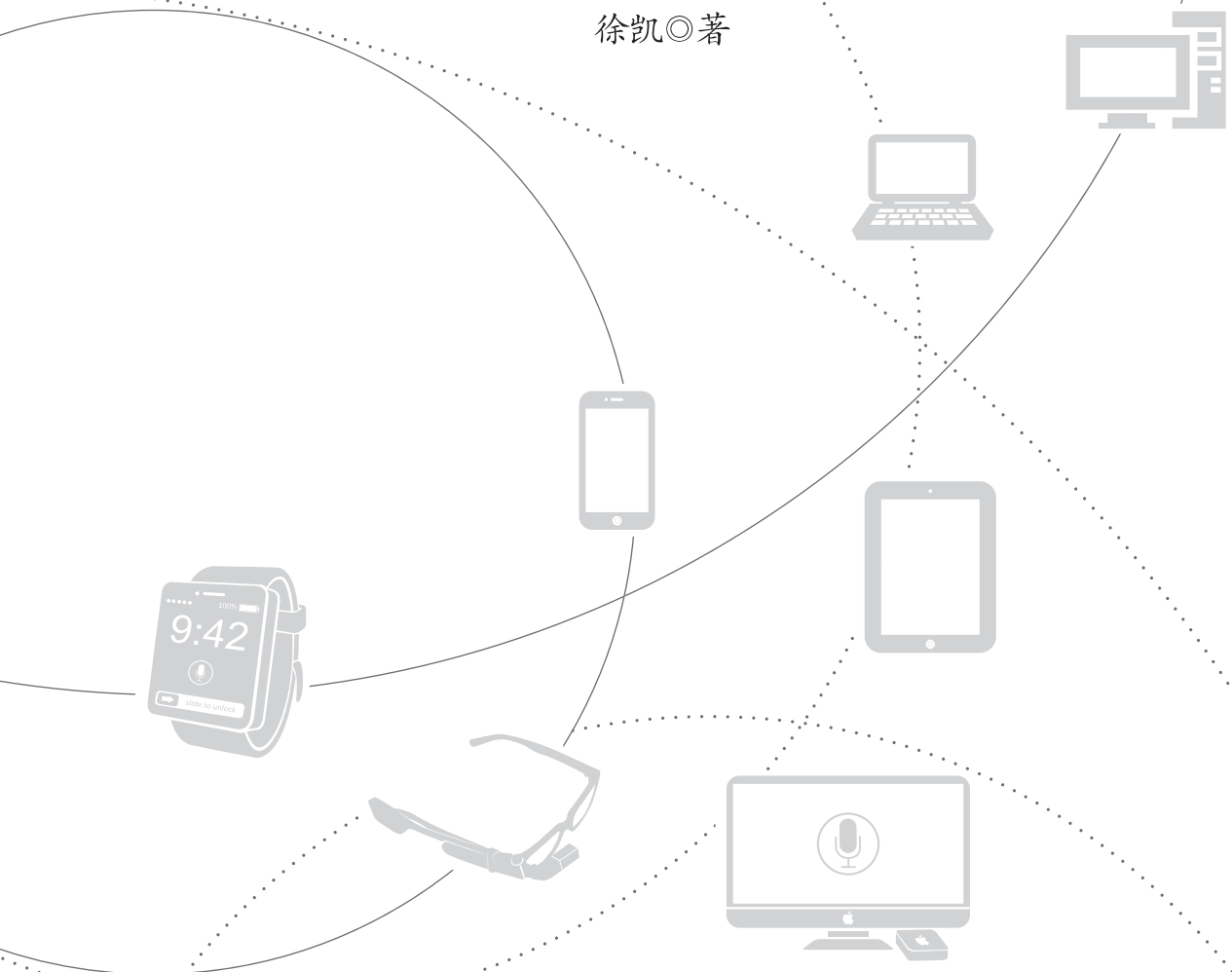
# 跨终端Web

徐凯◎著




電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

## 内 容 简 介

移动互联网不可阻挡地进入了我们的生活。作者将自己在百度和天猫期间的跨终端 Web 的开发实践转化为书中的技术方案和实现，呈现给各位读者。第 1 章提出了跨终端 Web 的概念以及实现跨终端 Web 的多重途径，第 2 章主要介绍 Mobile Web 的技术基础，第 3～7 章是全书的核心，按照开发流程组织逐步讲解了实现跨终端 Web 所需要的各类技术基础设施，第 8 章主要介绍了 Hybrid App 的发展历程、实现细节以及成熟的框架，第 9 章介绍的跨终端存储方案（Storage）是作者曾经的冠军作品，第 10 章完整介绍了如何通过脚本录制和回放来实现跨终端动作同步。

本书讲解深入浅出，通畅易懂，适合有一定 PC Web 基础，希望迅速了解 Mobile Web，致力于 PC 和 Mobile Web 技术融合的读者。



**图书在版编目（CIP）数据**

跨终端 Web / 徐凯著. —北京：电子工业出版社，2014.6
ISBN 978-7-121-23345-6

Ⅰ. ①跨… Ⅱ. ①徐… Ⅲ. ①网页制作工具 Ⅳ. ①TP393.092

中国版本图书馆 CIP 数据核字（2014）第 110704 号

策划编辑：张春雨
责任编辑：付　睿
印　　刷：北京丰源印刷厂
装　　订：三河市鹏成印业有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：15.25　　字数：270 千字
版　　次：2014 年 6 月第 1 版
印　　次：2014 年 6 月第 1 次印刷
印　　数：4000 册　　定价：55.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件到 dbqq@phei.com.cn。

服务热线：（010）88258888。

献给我的父母。

献给李玉北，我的第一位导师。

献给三七，我的第二位导师。

献给舒文，我在天猫的师兄。

献给所有百度和天猫时期的战友们。

# 推荐序

2012 年年底，我从云 OS 运营团队转岗到天猫前端团队，之前和手机硬件厂商、Mobile OS 及 App 市场的合作经历让我重新思考前端的未来和电子商务的技术方向。除了在天猫内部实践基于移动优先（Mobile First）的设计研发外，还和 InfoQ 合作把 Qcon 北京 2013 的前端议题定为“跨终端 Web”，于是就有了采访稿《从可编程到跨终端——QCon 北京 2013“跨终端的 Web”专题出品人鄢学鹃专访》[1]，阐述当时的认知和想法。现在“跨终端 Web”已经诞生一年半了，鬼道同学把这些想法和实践都编写成书实在让人惊喜和敬佩，我有幸成为这个成就的见证人，并再次有机会和大家谈谈跨终端 Web 的初心和思考。

1991 年诞生了 World Wide Web、HTTP 和 HTML，1995 年诞生了 JavaScript、Java 和 PHP，1996 年诞生了 CSS，1998 年 CSS 2.1 正式发布，1999 年 CDN 诞生且 HTTP 1.1 和 HTML 4.01 正式发布，2000 年 ECMA-262 3rd 正式发布，2001 年 IE 6 发布。作为 Web 开发者和前端工程师的我们应该对这些大事件相当不陌生，但如果你仔细看看这些年份就会发现我们所使用的绝大部分技术和方案都比 IE 6 老。在那个时期没有真正意义的前端，常见情况是美工切图，后端嵌套模板，也没有所谓的前端架构，前后端的发布机制几乎一样，那就是门户盛行的时代，Yahoo、新浪、搜狐和网易当道。

IE 6 是第一轮浏览器大战的胜者，垄断式的胜利直接导致 Web 基础设施至少 10 年的缓慢发展，标准的意义也就不那么明显了。其实 1994 年就有了 W3C，但民间为了解决 W3C 面对浏览器厂商不给力的混乱状况于 1998 成立了 WaSP（The Web

[1] http://www.infoq.com/cn/news/2013/01/qcon-beijing-web-interview

Standards Project），其中一个创始人于 2003 年发表了影响力巨大的著作《Design with The Standards（网站重构）》，当然在中国产生了估计作者都没有想到的额外影响，那就是出现了重构工程师。2004 年 Firefox 1.0、Gmail 和 Google Sugguest 发布，2005 年 AJAX 横空出世和 Google Maps 发布，2006 年 jQuery 和 YUI 发布。这些事件的到来显示出了低成本的 Web 在跨浏览器上的巨大威力和人机交互的显著改善，导致当时互联网 99%以上内容都是使用 Web 来呈现的，所以对于当时绝大部分人而言，WWW 就是互联网。巨大的需求导致 Web 开发的规模化和专业分工，这就是前端工程师开始出现并大量集结于以 Web 业务为主的互联网公司的原因。

我们可以称这个时代为前端工程师 1.0，比如在 Google、Facebook、Yahoo、阿里巴巴、百度、腾讯等公司中，前端团队小有规模且以 Web 开发为主，此时的前端工程师们大量使用 HTML、CSS、JavaScript 开发基于桌面浏览器的网页和应用。这个时代的前端工程师主要具备 3 个方面的能力，一是跨浏览器的兼容能力，需要理解渐进增强和优雅退化的思想，深入 Web 标准并结合环境数据制定 GBS（Graded Browser Support 浏览器分级标准）；二是富交互 Web 的开发能力，能够基于，甚至开发 JavaScript 库去实现人和机器的复杂互动；三是性能优化能力，实践表明前端占 Web 性能 80%，类似 CDN 和按需加载等各种性能优化方案深度地影响了前端架构和发布机制。

虽然今日市面上大部分前端工程师招聘条件还是基于前端工程师 1.0 的，但我们的环境自 2007 年 iPhone 发布起就在悄然却快速地变革，我们可能是家里唯一使用电脑的那个人，而手机在吃饭、走路、约会、上厕所等时都已经成为不可或缺的陪伴。当我还在云 OS 做运营时，便有一个令人震惊的发现：人类发展五千年到现在的最大共同点可能是每人都有移动智能设备。人们已经处于各种各样碎片化的智能移动设备上，这些设备有触摸屏、相机、麦克风、陀螺仪、加速计等特性，使得人同机器交互的方式发生了巨大的革命，更重要的是它们都和网络连接，没有了离线状态。于是，当一个人在大街上看到奇闻异事随手用手机拍下来并发到微博上，TA 的朋友可能在手机、平板电脑或普通电脑的微博上看到并转发，也可能被 TA 的朋友转到微信或 IM 等，TA 朋友的朋友同样可能在手机、平板电脑或普通电脑的另一种软件上看到这个信息，甚至信息被电视媒体或平面媒体发现并造成更大范围的传播，更多 TA 不认识的电视或平面媒体通过扫描报道时的二维码又找到 TA 的微博，这一切

可能在瞬间完成。这个小场景告诉我们人和信息是在智能终端（Phone、Pad、Desktop、TV 等设备）、软件终端（浏览器、SNS、IM 等应用）和传统终端（电视、广播、平面等媒体）间交叉流动的，这是一个去中心的网状结构，也是互联网的本质特性，所以跨终端不仅仅是跨越设备（device），更是跨越人机交互的场景入口（end）。

面对这种终端碎片化的潮流，前端工程师怎么办？解决方案就是基于最重要的前端开发思想渐进增强和优雅退化得出的移动优先：一是毫无疑问绝大部分用户已经或正在成为智能设备用户，我们要为 80%的目标用户服务；二是专注于核心业务需求，人的本性、业务本质和商业模式本质基本上不会随着终端改变而改变，所以相同业务在手机、平板、桌面和电视上呈现的本质和商业模式不会有不同，小屏幕终端是我们重新思考业务本质和核心人机交互流程的机会，其挖掘出的本质会改变其他终端；三是针对未来人机交互，现在移动设备引领人机交互的变革潮流，通过必备特性虚拟或增强现实，并逐步引入到桌面和电视等设备中。所以，选择哪个具体技术方案，是响应式 Web，还是服务端响应式 Web（URL 不变，通过服务端在不同设备展现不同模板），还是多个 URL Web（不同的终端，不同的 URL），还是 Hybrid 应用（Native 和 Web 混合使用，比如 iOS App Store 一直就是引用壳里面套个网站），还是 Native 应用，是依据业务本质、人性需求和人机环境趋势来综合判断的。由于 Web 的本质特性就是低成本跨平台但对设备先进特性支持不够，而 Native 应用能够充分利用硬件先进特性但受限于系统平台，导致开发者没有发布能力，所以大多数情况下，可以把跨终端 Web 作为默认选择。

在跨终端的时代，渐进增强和优雅退化依旧是最重要的前端开发思想，但前端工程师们不能像以前一样仅仅固守在 Web 上，Web 和客户端应用的融合已经成为必然，Web 从页面（page）到应用（application）反映了人机交互革命带来新的体验趋势，客户端和动画开发成为了前端工程师的基本功。移动优先的跨终端解决方案核心是一套数据有多个高品质低成本展现方式，这促使前后端分离成为必需，前端工程师不仅仅要关心客户端环境也要关心服务端环境，所以 GBS 需要升级到 GTE（Graded Target Environments，分级目标环境），工程师更关注端到端的数据接口约定（比如正文中的 IF 就是这种思想下的一种实践方案），这会彻底改变前端的开发方式、架构和发布机制，也会导致前端团队快速膨胀直到前后端比率趋于一致。这是一个巨

大的挑战，也是一个前所未有的机会，更是时势的要求。我把这个时代叫作前端工程师 2.0。

这篇推荐序被我拖延了很久很久，结尾之时恰逢鬼道同学入职天猫一周年，就拿老乔的话来 zhuangbility 一下：“你如果出色地完成了某件事，那你应该再做一些其他的精彩事儿。不要在前一件事上徘徊太久，想想接下来该做什么。”这应该是最好的礼物之一。

三七

2014.4.8

# 前言

移动互联网不可阻挡地进入了我们的生活。笔者将自己在百度和天猫期间的跨终端 Web 的开发实践转化为书中的技术方案和实现，呈现给各位读者。

本书大纲来自于笔者 2013 年 7 月在 D2 上的主题分享“移动优先的跨终端 Web”[2]，2013 年 11 月 W3CTECH 2013 上做了第二次分享[3]。

## 面向对象

本书适合有一定 PC Web 基础，希望迅速了解 Mobile Web，致力于 PC 和 Mobile Web 技术融合的读者。

本书的第 1 章、第 3 章、第 5 章面向所有的读者，即使您对 Web 技术毫无了解，仍可以顺利阅读下来。本书其他章节假定您对 PC Web 前端技术已经有所了解，知道 HTML、CSS、JavaScript 这些名词的含义，并且动手写过一个页面。

## 章节简介

第 1 章阐述了移动互联网的现状和遇到的问题，提出了跨终端 Web 的概念以及实现跨终端 Web 的多重途径，破除了“唯 Media Query 论”，并引入移动移先的概念，丰富跨终端 Web 的内核。

第 2 章从对比 PC Web 的角度介绍了 Mobile Web 的技术基础，并对远程调试、兼容

---

[2] http://luics.github.io/demo/d2/

[3] http://luics.github.io/demo/cew-w3ctech-1311/
在线视频：http://www.imooc.com/course/video/?mid=535

性等开发问题做了详尽的阐述。

第 3～7 章是全书的核心，按照开发流程组织，逐步讲解了实现跨终端 Web 所需要的各类技术基础设施，包括：

- 基准，给出了调试和测试过程中的基准，确定调试和测试的范围。
- 检测，探讨如何构造一个为全站服务的终端属性检测工具。
- 接口，探索实现流程复用的途径及端到端的接口规范（IF）。
- 定位，分别介绍了基于 Hash 和基于 History API 的跨终端定位方案。
- 预览，介绍了实现跨终端预览的方案，并给出一个简化的实现。

第 8 章介绍了 Hybrid App 的发展历程、实现细节以及成熟的框架，并详细分析了以传感器为核心的 Device API。

第 9 章的跨终端存储方案（Storage）是笔者参加“2013 Kissy Gallery 组件大赛”时的冠军作品。

第 10 章完整介绍了如何通过脚本录制和回放来实现跨终端动作同步。

# 感谢

感谢三七（鄢学鹃），“跨终端 Web”这个概念最早由三七提出，感谢三七为本书写序以及在本书写作过程中给予的支持。三七是位充满智慧的前端架构师和优秀的管理者，从他那得到的启示远不止跨终端 Web，在此一并谢过。

感谢舒文（舒文亮），他为本书第 4 章“检测”写了初稿。舒文带领跨公司的团队实现了服务于全阿里的多终端检测（MED）系统。

感谢惜朋（沙峰），他为本书第 10 章“动作同步”撰写了“脚本录制和回放”的初稿，并实现了其中的 Demo。

感谢闭月（吴燕云）和蔡伦（盛柳钦），他们为本书第 7 章“预览”提供了 Focus 截图和服务端代码。闭月和蔡伦的家庭新成员即将诞生，在此祝福小生命为他们的生活带来更多的欢乐。

感谢 IF 项目虚拟团队的蒋壮（王跃乐）、劳谦（董松洁），他们分别为 IF 在测试和后端领域内的扩展做了开拓性的工作。

鬼道 于 2014-02-23

# 目录

# 1 跨终端 Web

## 1.1 终端 VS.设备

当移动页面只运行于独立浏览器中时，“终端”和“设备”（图 1-1）这两个名词在多数场景下是可以相互替换的。现在移动页面不仅仅运行于独立浏览器中，也运行于众多 App 的 WebView[1]中。终端和设备的概念出现了微妙的变化：页面可以运行于同一设备的不同终端下。本书取名为“跨终端”而不是“跨设备”也正是适应了这个变化。

“跨终端”中的“跨”强调了在多个终端上的信息传递、相互协作等动态内涵。跨终端希望从移动优先的理念出发，从设计、产品、开发都能够先做出移动版本再层层累加，这是理念上的“跨”；另一方面，对于用户而言，看到的页面地址永远都只有一个，不同终端间的差异应该对用户透明，这是现实的“跨”。

图 1-1 移动设备

[1] Android 中称为 WebView，iOS 中称为 UIWebView，后续章节简称为 WebView；WebView 就好比是嵌入 App 的迷你浏览器，实际上这个迷你浏览器和大部分独立浏览器一样都使用 WebKit 内核。

## 1.2　一个贯穿全书的例子

为了让“跨终端 Web”更加容易理解，本书多个章节会反复提到这样一个例子：

> 在PC上分享了一个链接A,好友在移动终端上看到这个分享并单击链接A，期望看到一个正常显示并可操作的页面，反之亦然。

例子中的链接对应的可以是商品购买页、博客页面等任何形式的页面，例子中的“操作”是指操作页面的核心功能，如购买、发表评论等。在国内，2007 年以前，大部分情况下，你会看到类似图 1-2（左图）显示的页面。页面加载速度非常慢，诸多功能无法使用，同时这样的展示效果也让人没有意愿继续操作。

为了解决上面的问题，该站点专门为移动端设计了一个页面 B（图 1-2 右图），并且用户在移动端访问链接 A 时会自动跳转至页面 B，反之亦然。另外一个更好的做法是只使用链接 A，服务器端根据 UA 判断返回最适合终端展现的页面。


图 1-2　传统页面与跨终端页面对比

本书的雏形是笔者在 2013 D2 分享的主题“移动优先的跨终端 Web”[2]，当时为了更形象地展示一些技术方案，我们做了一个 Demo[3]。这个 Demo 就可以使用同一个链接在 PC、平板、手机端访问，并展示出最适合终端的界面。

## 1.3 后续章节

看起来跨终端问题就这样被彻底解决了，本书到此也就该结束了？

对于一个只有单个页面的简单站点而言，实现跨终端要做的事情无非是再考虑多做几个移动端版本，或者直接用响应式布局一次性解决所有终端的展示问题，但是对于构建一个庞大的站点而言，跨终端之路才刚刚开始。

1. 本章的剩余章节将会继续讨论跨终端 Web 中的移动优先理念，以及跨终端 Web 技术方案的探索。
2. 第 2 章，对比 PC Web 技术补充 Mobile Web 中涉及的基础技术。
3. 第 3 章，给出了调试和测试过程中的基准。
4. 第 4 章，终端检测探讨如何做一个为全站服务的终端属性检测。
5. 第 5 章，探索实现流程复用的途径及端到端的接口规范。
6. 第 6 章，跨终端定位的技术方案。
7. 第 7 章，介绍了如何实现一个终端预览。

第 3～7 章是全书的核心，按照开发过程涉及的技术方案组织，如图 1-3 所示。第 8 章开始介绍了和跨终端相关的诸多技术：Hybrid App、跨终端存储、跨终端动作同步，其中的跨终端存储方案（Storage）是笔者参加“2013 Kissy Gallery 组件大赛”的冠军作品[4]。

[2] http://luics.github.io/demo/d2

[3] http://luics.github.io/demo/d2-demo/demo/sub.html

[4] Kissy 是阿里的 JavaScript 库 http://docs.kissyui.com/，阿里巴巴集团内部的“Kissy Gallery 组件大赛”，2013 年参赛组件达到 75 个。

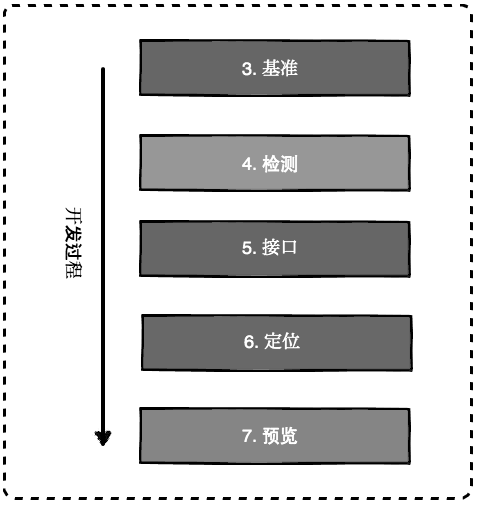



图 1-3　本书第 3～7 章的组织结构

## 1.4　移动优先

1.3 节内容提到了移动优先，这个概念从 2009 年 11 月提出到现在也不过 4 年多的时间[5]，但是已经风靡全球。移动优先的概念涵盖了：移动流量暴增、聚焦业务本质、人机交互扩展，其中的“聚焦业务本质”是核心。

### 1.4.1　移动流量暴增

在国内，10 多年前已经开始使用功能手机（相对智能手机而言）在 2G 网络环境下浏览网页，由于网络、手机性能的局限当时普遍使用 WAP 这种过渡性方案；随着 2007 年的 iPhone 和 2008 年的 Android 智能手机的相继出现，移动端的上网人数急剧增加，统计显示 2012 年年底中国手机网民总量 4.2 亿，占到整体网民数量的 75%[6]；从增长率上看 2008 年达到了峰值 133%（和 iPhone、Android 相继问世有关），直到 2012 年增长率仍然维持在 18%，如图 1-4 所示。

[5] http://www.lukew.com/ff/entry.asp?933

[6] 中国互联网中心（CNNIC）：《中国移动互联网发展状况报告》2013 年 4 月。


图 1-4 CNNIC：中国手机网民规模及增长率[7]

2012年7月移动端平均上网时长首次超过PC端，2013年7月这个差距已经达到29%，并且趋势显示差距仍在进一步拉大[8]，如图 1-5 所示。

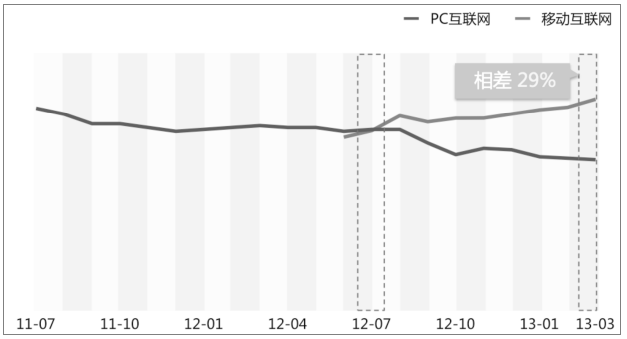



图 1-5 移动端和 PC 端人均上网时长

国外的情况也大致如此，iPhone 的出现让 AT&T 的移动端流量在 2007—2009 年间增长了 4932%。所谓移动优先，首要考虑的就是让公司的业务对这部分庞大的人群可用。

## 1.4.2 聚焦业务本质

大部分智能手机屏幕宽度在 320 像素到 480 像素之间，高度一般不超过 600 像素[9]，对于这样的屏幕而言能做的就是展示业务的核心内容。这种环境下反而能够促使我

7 中国互联网中心（CNNIC）：《中国移动互联网发展状况报告》2013 年 4 月。

8 百度《移动互联网发展趋势报告》2013 年 Q1。

9 参考本书“基准”一章。

们更加聚焦业务的本质。

移动优先不仅仅指开发，更应该是组织架构、产品规划、设计、开发这几个层面的移动优先。因为只有这样，开发层面的移动优先才能最终得以推行并取得预期的成果。图 1-6 是一个典型的电商类网站的跨终端站点架构，与很多公司现行架构不同之处在于：同一个团队需要同时负责 PC 端和移动端，这样的组织架构是保证跨终端高效推行的基础，因为同一个业务无须因为终端的不同而分给不同的团队维护，反之会增加沟通成本，造成资源的浪费。

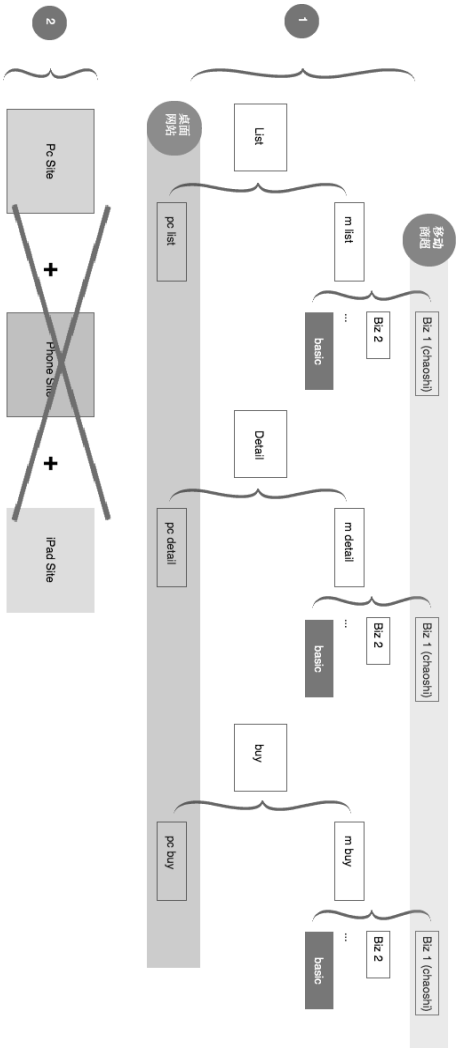



图 1-6　跨终端站点架构

### 1.4.3 人机交互扩展

相对 PC 上鼠标或触摸板等点选设备的体验而言，移动设备有更丰富、先进的人机交互体验。产生这些体验的感应器至少包括我们熟知的触摸屏、音频、视频、定位、陀螺仪、光线感应器、压力感应器等各类先进的传感器。

尽管智能手机的初期，如 Symbian、Black Berry，仍以物理键盘作为主要的输入设备，但是随着 iPhone 和 Android 的普及，现在的智能手机几乎都配有触摸屏，想再找一款含有物理键盘的智能手机反而变得困难了。触摸功能让移动设备（尤其是手持设备）可以做得更小，绝大多数场景的操作也更加便捷；同时触摸（尤其是手势）相对键盘输入降低了操作门槛，更容易被用户接受。

移动设备内置的 GPS 模块可以提供当前位置信息。基于位置的服务（LBS，Location Based Service）如今已经影响到了移动互联网的诸多方面，如搜索、地图、导航、电商、娱乐、团购、社交、即时通信，持续地改善着普通人的生活。尤其是 GPS、加速计、陀螺仪、磁力计的结合就能实现传统导航仪所具有的功能，而导航的功能又不仅仅局限于车载导航领域，甚至可以用作个人运动的监控。

随着 Siri 的横空出世，之前未被广泛利用的语音输入也进入人们的生活。触摸能够降低很多操作的门槛，但对于文字输入而言并没有太多帮助；语音输入则在这方面做得非常出色，配合较高的语音识别率，已经能够做到使用语音输入替代键盘输入。各类“语音助手”除了提供简单的语音转文字的服务，还提供各类衍生的服务；同时诸如浏览器一类的 App 也集成了语音输入的功能，如图 1-7 所示。


图 1-7 语音输入

“摇一摇”是基于加速计开发的最有趣的功能之一，已经被广泛应用于各类场景中；光线感应器可以帮助屏幕自动调节亮度，甚至在 CSS Media Query 最新草案中还引入了 luminosity 特性[10]，页面根据环境改变皮肤的功能或许很快就能实现了。

移动端精彩之处在于谁也不知道明天会出现个什么新的交互方式。或许终有一天拿着手机，即使不说话对方也能听到你的想法，因为手机已经内置了脑电波识别模块。

### 1.4.4 再说书名

笔者以“移动优先的跨终端 Web”为题做过多次公开演讲[11]，为何本书没有继续使用该标题？最初确实是为了简化标题，但细想来移动优先的理念真正融入到产品、设计、开发环节中的时候跨终端 Web 就成功了，那时也无须再去提及“移动优先”，所以从一开始我们就用“跨终端 Web”为名或许更为合适。

## 1.5 不只是响应式

一提到跨终端，第一反应往往就是响应式布局。这至少说明两点：首先，响应式本身与跨终端之间有着某种本质的联系；其次，人们误以为跨终端和响应式是同一件事。本书所述的“跨终端 Web”是最终希望达到的目的，而达到这个目的的手段有很多，响应式仅仅是其中的一种方式而已。这些方式至少包括：

1. 响应式
2. 多站点
3. 多模板
4. 多平台

### 1.5.1 响应式

移动互联网的诞生尤其是智能手机普及之后，各种终端（图 1-8）呈现爆炸式增长，在面对如此多的终端时的确会慌了手脚——传统 PC 上一套布局远远不能满足现在

[10] http://dev.w3.org/csswg/mediaqueries4/#mf-environment

[11] http://luics.github.io/

的需求。为应对这个危机Ethan Marcotte在2010年5月发表的*Responsive Web Design*[12]（其搜索热度见图 1-9）中首次提出了响应式网页设计这个词（下文简称响应式）。响应式的核心概念是一个站点能够兼容多个终端，而不是为每个终端做一个定制版本。这个概念正是为解决移动互联网而诞生的，所以提到跨终端对大部分人而言自然会想到响应式。由于响应式很好地满足了移动互联网的需求，迅速风靡全球。


图 1-8　各种终端

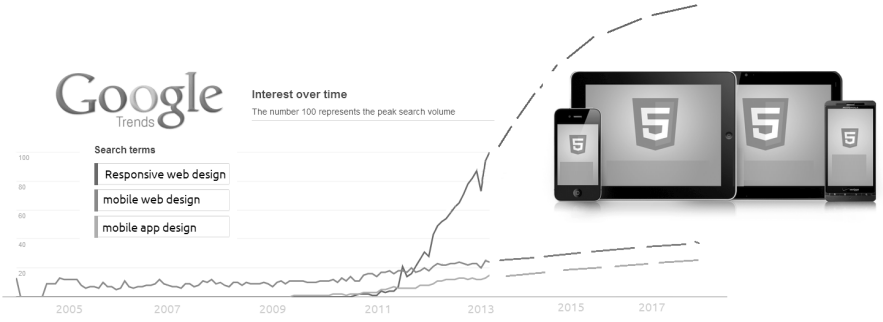



图 1-9　Responsive Web Design 搜索热度

Ethan Marcotte 在他的文章里提供了一个“福尔摩斯历险记”的例子[13]，在不同的屏

[12] http://alistapart.com/article/responsive-web-design

[13] http://alistapart.com/d/responsive-web-design/ex/ex-site-flexible.html

幕宽度下呈现出不同的展现样式，非常形象地诠释了响应式的理念。可以看到：

- 在宽度大于 1300px 时，整页分为左右两列布局，6 个人物头像一行展示出来，如图 1-10 所示。


图 1-10　宽度大于 1300px

- 当宽度在 600px 和 1300px 之间时，整页仍为左右两列布局，不同的是右侧人物头像 2 行展示，并且图片是自适应的，如图 1-11 所示。


图 1-11　宽度在 600px 和 1300px 之间

- 当宽度在 400px 和 600px 之间时，整页变为一列布局，同时人物头像继续 2 行显示并且更小，如图 1-12（左）所示。
- 当宽度小于 400px 时，人物头像变为 3 行，如图 1-12（右）所示。


图 1-12　宽度在 400px 和 600px 之间（左），宽度小于 400px（右）

这个例子涵盖了目前主流的 4 类终端：PC 大屏、PC 小屏（笔记本）、平板电脑、手

机，如图 1-13 所示。


图 1-13　主流的 4 类终端

更多响应式的技术细节不属于本书的讨论范围。响应式的入门介绍推荐这篇《自适应网页设计（*Responsive Web Design*）》[14]文章。这里再推荐一个现在非常流行的响应式 Web UI 库 Foundation[15]，Foundation 是一个易用、强大而且灵活的框架，用于构建基于任何设备上的 Web 应用。提供多种 Web 上的 UI 组件，如表单、按钮、Tabs 等。也可以参考这篇《你可以构建复杂的响应式 Webapp》[16]文章，它从工程实践角度讲解了如何结合 Foundation 以及第三方工具库构建一个较为复杂的响应式 Webapp。

## 响应式的瓶颈

对于DOM结构异常庞大复杂的页面而言，响应式不能解决移动端DOM冗余的问题，JavaScript 脚本冗余也是一个问题；从工程实践来看，单个复杂响应式页面的维护成本并不比单独维护多个版本的页面成本低，并且由于响应式存在的内在耦合性，这个维护成本在复杂页面频繁更新时反而更高。响应式本质上是依靠 CSS 处理展现层

[14] http://www.ruanyifeng.com/blog/2012/05/responsive_web_design.html

[15] https://github.com/zurb/foundation

[16] http://adioso.com/blog/2013/06/responsifying-adioso/

面的差异，从笔者所接触到的工程实践来看，移动端和 PC 端存在着不仅仅是展示上的差异，在交互形式上的巨大差异也会导致 DOM 结构上的差异，这种差异已经远远超出 CSS 所能控制的范围。

### 1.5.2 多站点

移动互联网初期（WAP 还是主流），国内较大的几个 PC 门户站点推出了所谓的“极简版”，智能手机问世后又出现了“炫彩版”，平板电脑问世后有了“HD 版”，甚至还有了专门针对 iPhone、iPad 的特殊版本。再加上已有的 PC 版，通常一个站点 example.com 的首页就有了这么一串地址：www.example.com、wap.example.com、m.example.com、hd.example.com、iphone.example.com、ipad.example.com……

设想一个 PC 主业务流程有 10 个二级子域的站点，在跨终端场景下恐怕得维护 50 个以上的二级域名，这样会加大服务器端维护跳转的成本，对用户而言同一个页面出现多个域名也是很困惑的一件事情。

### 1.5.3 多模板

多模板是响应式和多站点相结合的一种方案。以 1.2 节中提到的 Demo 为例，Demo 使用了两套模板：Tiny、Normal。Tiny 模板支持宽度 480px 以下的屏幕，Normal 模板支持 480px 以上任何屏幕宽度，这得益于其使用的 AutoResponsive 响应式流布局组件[17]。

多模板的优点在于一个页面只有一个 URL，无须服务器端复杂的 URL 映射规则和终端检测等手段进行跳转。虽然解决了响应式中 DOM 冗余的问题，但是由于单个页面存在多套模板，还需要在模板动态加载和首次服务器渲染等环节进行优化。图 1-14 是移动端模板，图 1-15 是平板电脑和 PC 共用一套模板。

[17] https://github.com/xudafeng/autoResponsive


图 1-14　移动端模板


图 1-15　平板电脑和 PC 共用一套模板

### 1.5.4　多平台

Native App 的确也是实现跨终端 Web 的一种途径，如图 1-16 所示。它的优势在于：更好的性能、更丰富的系统级功能的调用、标准的发布渠道（通常是应用商店）。

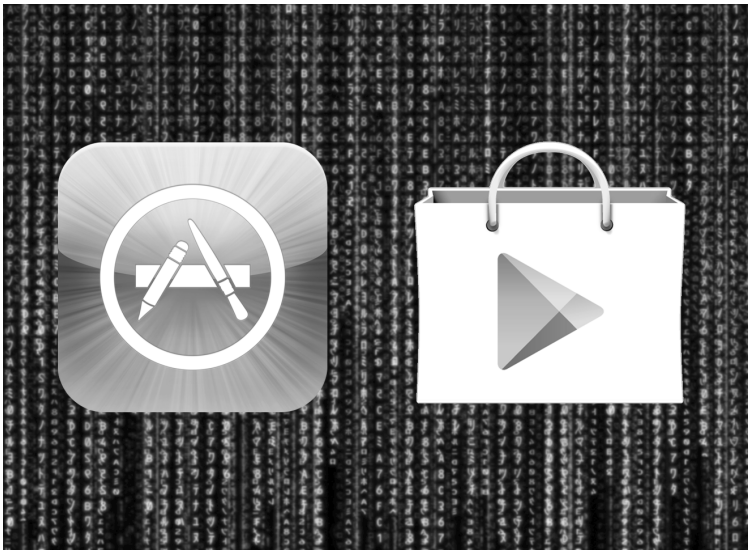


图 1-16　Native App

Native App 的劣势也是非常明显的。

**缺陷 1，发布成本高**。iOS 上的发布几乎都通过 App Store，审核周期一周至数周不等，这对需要及时更新内容的 App 而言是致命的；对于 Android 平台而言，应用商店众多也是个头疼的问题，因为一份更新可能需要涉及超过 10 个主要的应用商店，无论是维护的成本还是发布周期都让人吃不消。以国内某知名 App 为例，保证平均每个版本更新率达到 80%要 6 个月。

其实对于缺陷 1，目前 Native App 使用 WebView（App 内置浏览器）维护需要及时更新的内容，从代码量上看 Facebook App 中 Web 代码（HTML、CSS、JavaScript 等）部分占到 90%，而 Linkin App 更是达到了 95%。

**缺陷 2，开发成本高**。iOS 和 Android 的开发成本明显高于 Web，保守的估计是 1.5∶1。可以将浏览器看作是特殊的 Native App，而 Web 是在这个特殊 App 之上构建的，Web 开发中已经将一些底层细节屏蔽，所以其开发成本会更低也是很自然的。最为

关键的是，Web 不存在需要为不同的平台维护一套 App 这个问题。

**缺陷 3，潜在的风险**。iOS 上的发布严重依赖于 App Store，而政策上的风险可能是致命的，例如某电视台集中对苹果公司的曝光事件；Apple 的策略可能会给 App 带来严重的影响，例如 iOS 上的浏览器无法使用第三方浏览器内核，Google 同样可能有类似的问题；Android 平台上应用商店参差不齐，也可能导致意外的风险。

Hybrid App 是多平台 App 的一种改进形式，由于它的重要性本书将会专列一章进行讨论。

## 1.6　解决方案

继续 1.5 节的话题，我们的跨终端 Web 到底该用哪种技术方案呢？虽然响应式是一个不错的选择，但多站点仍是实现跨终端最为直观的方式，并且在复杂的大型站点环境下仍然是首选的方案。直到今天我们知道阿里系的天猫、淘宝，百度和腾讯的大部分应用也仍然是在使用多站点的方案。不过大型站点往往不会只是使用单一的方案，以 BAT[18]为例，在整体使用多站点的方案下，部分应用可能也会使用响应式，同时 BAT 也维护着数量庞大的 Native App。

[18] BAT（Baidu Alibaba Tencent）是互联网上对 3 家公司的简称。

# 2

# Mobile Web

回顾 PC Web 上用到的基础技术：HTML、CSS、JavaScript，如图 2-1 所示。HTML 负责页面结构，CSS 负责页面样式，JavaScript 为页面提供了行为能力。以房子为例，HTML 好比是骨架，CSS 是内外装饰，JavaScript 就是房子内所有的功能（照明、取暖等）。Mobile Web 中的基础仍是 HTML、CSS、JavaScript，不同的是，Mobile 浏览器内核几乎都使用了较新版本的 WebKit，因而在这 3 个方面 Mobile Web 有了诸多的扩展。

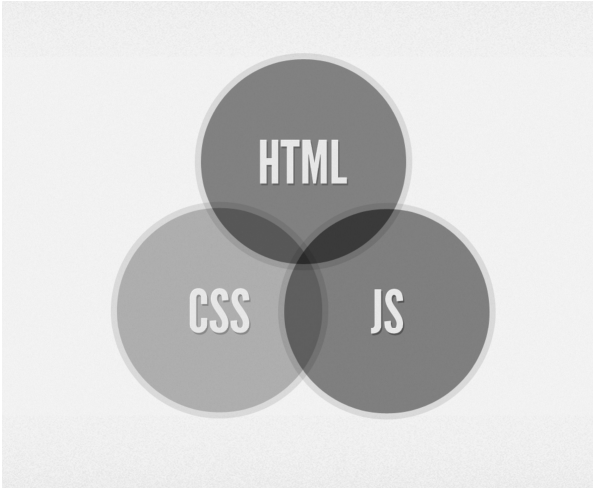



图 2-1　Web 基础

今天移动端的崛起速度之快远超出商业集团的应变能力，但是 Mobile Web 也仍然还是 Web，就像 HTML5 只不过是 HTML 的一个版本一样。在笔者看来，Mobile Web 和 PC Web 在技术层面上并无本质的差异。

# 2.1　HTML5

Mobile Web 与 PC Web 在基础技术上的差异主要体现在 HTML5 的广泛应用以及本书“Hybrid App”章节会提到的 Device API。这里的 HTML5 ~= HTML5 + CSS3 + JavaScript。

首先来看下 HTML 的发展历程。

1. 1999 年，HTML 4.01 发布。
2. 2000 年，W3C 发布了 XHTML 1.0。
3. 2004 年，WHATWG 提出 Web Applications 1.0[1]，这是 HTML5 草案的前身。
4. 2007 年，W3C 接纳了 Web Applications 1.0，并成立了新的 HTML 工作组（合并 XHTML 团队）。
5. 2008 年，第一份 HTML5 正式草案公布[2]。
6. 2012 年，W3C 和 WHATWG 决定独立维护 HTML5 标准，之后 W3C 的 HTML5 标准将会是 WHATWG 的一个快照[3]。
7. 2014 年，HTML5 成为 Recommendation（推荐规范）[4]。

HTML5 的新特性[5]体现在图 2-2 显示的 8 个部分，HTML5 希望创建一个有本地存储、富客户界面、高效网络 I/O 的 Web App。


图 2-2　HTML5 新特性

[1] http://www.whatwg.org/specs/

[2] http://www.w3.org/TR/2008/WD-html5-20080122/

[3] http://lists.w3.org/Archives/Public/public-whatwg-archive/2012Jul/0119.html

[4] http://www.w3.org/2011/05/html5lc-faq.html

[5] http://www.w3.org/html/logo/#the-technology

1. 语义（Semantic），新增 header、footer、nav、fig 等含有语义的标签，以及一系列含有语义的标签属性。
2. 离线&存储（Offline&Storage），主要包括 Local Storage、Indexed DB、File API。
3. 设备访问（Device Access），定位信息已经广泛应用，其他还有视频、音频流（如语音输入），移动设备的传感器（如方向传感器）。
4. 网络连接（Connectivity），增加 Web Socket、服务器数据推送。
5. 多媒体（Multimedia），增加 video、audio 标签，提供原生的视频、音频访问。
6. 图形接口（GDI），增加 canvas 标签，提供 2D、3D GDI，现已有第三方的 WebGL 可以提供 3D 加速渲染。
7. 性能&整合（Performance & Integration），Web Workers 实现脚本后台运行，并提供前后台交互接口，XMLHttpRequest 2 提供更好的网络 I/O。
8. CSS3，目前仍在开发之中，主流浏览器已经支持其中部分特性，Mozilla Dashboard[6]是更完整的 CSS3 Demo。

“HTML5 Presentation”[7]是使用 HTML5 开发的一个站点，详尽介绍了 HTML5 诸多特性，同时也是不错的入门 Demo。本章不再累述 HTML5 的所有新特性，接下来将着重介绍 HTML 和 JavaScript 中非常有 Mobile 特色的内容。

# 2.2 HTML

## 2.2.1 移动页面模板

先给出一个移动页面的模板[8]，如代码 2-1 所示。

**代码 2–1 移动页面模板**

```
<!DOCTYPE html>
<html>
```

[6] https://mozillademos.org/demos/dashboard/demo.html

[7] http://slides.html5rocks.com/

[8] 本书所有代码均可从此处获得 http://luics.github.io/cew/index.html。

```
<head>
    <meta charset="utf-8"/>
    <title>Mobile Page Standard Template</title>
    <!-- 纠正：user-scalable=no 是需要的，发现在某些设备上仅有 initial 和 maximum 仍然无法阻止缩放 -->
    <meta
content="initial-scale=1,maximum-scale=1,user-scalable=no,width=device-width" name="viewport"/>
    <!-- touch-icon 在 iOS 中用于桌面图标-->
    <link
href="http://img01.taobaocdn.com/tps/i1/T1zo51XxXfXXXeOHro-144-144.png" rel="apple-touch-icon-precomposed"/>
    <!-- 从桌面 icon 启动 iOS Safari 是否进入全屏状态(App 模式） yes | no-->
    <meta content="yes" name="apple-mobile-web-app-capable"/>
    <!-- iOS Safari 全屏状态下的状态栏样式  default | balck | black-translucent-->
    <meta content="black-translucent" name="apple-mobile-web-app-status-bar-style"/>
    <!-- iOS 设备上禁止将数字识别为可点击的 tel link -->
    <meta content="telephone=no" name="format-detection"/>
    <style type="text/css">
        body {
            margin: 0;
            background-color: #eee;
        }
        .header {
            height: 80px;
            background-color: #b00;
        }
        .body {
            width: 320px;
            margin: 0 auto;
            padding: 10px 0;
            background-color: #aaa;
        }
        .footer {
            height: 80px;
            background-color: #fff;
        }
    </style>
</head>
<body>
```

```
<div class="header">移动页面标准模板</div>
<div class="body">
    请查看源码
    <br/>
    1234567
</div>
<div class="footer">页尾</div>
</body>
</html>
```

这个移动页面模板和 PC 页面的差异在于 head 部分多了一些特殊的属性，接下来的章节将详细介绍这些属性。

## 2.2.2 Viewport

```
<meta content=" width=device-width,initial-scale=1,user-scalable=no,
    maximum-scale=1 " name="viewport"/>
```

上面的例子是 Mobile 页面使用 Viewport 的通用方式，也是本书推荐的方式。大意是：页面 CSS 计算时使用的宽度根据设备给定值自适应、初始不缩放、不允许用户缩放、最大缩放因子为 1。

Viewport 最初是 iOS Safari 的私有属性，以下属性定义摘自 Safari Developer Library[9]。

1. width：控制 Viewport 的大小，单位是 px，可以指定一个数值或常量“device-width”；iOS Safari 上默认值为 980，取值范围为 200～10000。
2. height：和 width 相对应，常量为“device-height”；iOS Safari 取值范围为 223～10000。
3. initial-scale：初始缩放比例。
4. maximum-scale：允许用户缩放到的最大比例。
5. minimum-scale：允许用户缩放到的最小比例。
6. user-scalable：用户是否可以手动缩放。

Viewport 还有另外一种常用的写法：

[9] http://developer.apple.com/library/safari/#documentation/appleapplications/reference/SafariHTMLRef/Articles/MetaTags.html

```
<meta content=" width=device-width,initial-scale=1,maximum-scale=1"
    name="viewport"/>
```

这种写法少了“user-scalable=no”。虽然“initial-scale=1，maximum-scale=1”可以保证在大部分情况下用户不可进行缩放，实测在某些 Android 设备的 WebView 上会出现用户仍可进行缩放操作的情况，所以推荐加上“user-scalable=no”。

上节标准模板中，元素“div.body”设置了宽度 320px 居中，这么做可以避免自适应宽度带来的开发成本，缺点是在一些 Android 浏览器中留下两边的空白；解决两边留白的简单办法是让页面头尾充满屏幕宽度。

### 2.2.2.1　固定 width

Viewport 中可以设置“width”为固定值，如代码 2-2 所示。

**代码 2–2　Viewport 设置固定 width**

```
<meta content=" width=320" name="viewport"/>
```

这种写法主要存在两个问题：一是横竖屏切换时图文变大，二是不同设备上的展现效果也有图文大小的差异，如图 2-3 所示。其实两个问题也可以看作是一类问题，都是因为物理宽度变化时 Viewport 宽度是固定值所致，自然图文的展现也会有差异了。这种差异在大部分时候是希望避免的，但是在需要图文充满屏幕的情况下，这种写法反倒合适。


图 2-3　Viewport width=320 横竖屏

### 2.2.2.2　动态载入

某些场景下会需要动态载入 Viewport（见代码 2-3），以便让原本不支持 Viewport 的页面能够更好地呈现在 Mobile 浏览器或 WebView 中，如图 2-4 所示。

代码 2–3 viewport 动态载入

```
function addViewportSupport () {
    var viewport = document.createElement ('meta') ;
    viewport.name = 'viewport';
    viewport.content = 'initial-scale=1,maximum-scale=1,user-scalable=no,
width=device-width';
    document.body.appendChild (viewport) ;
}
```


图 2-4 Viewport 动态载入前（左图）后（右图）

### 2.2.2.3 更多

Viewport 在 Mobile page 的开发中可能很不起眼，因为参照上面推荐的写法可以很好地运行在绝大部分设备的浏览器和 WebView 中，无须过多关注背后的深意。但笔者觉得理解了 Viewport 大有裨益，推荐继续阅读 MDN 上的这篇《在移动浏览器中使用 Viewport 元标签控制布局》[10]文章；另外，来自 quirkmode 的《Viewport 双城记》上下篇[11]深入揭示了 Viewport 背后的故事，也是笔者目前遇到过的对 Viewport 介绍

[10] https://developer.mozilla.org/zh-CN/docs/Mobile/Viewport_meta_tag

[11] 第 1 篇 http://www.quirksmode.org/mobile/viewports.html；第 2 篇 http://www.quirksmode.org/mobile/viewports2.html。

最为详尽深入的文章。

最后以《Viewport 双城记》中对 Viewport 属性产生原因的描述结束这一节。最初，大部分 Mobile 浏览器上的页面展示效果如图 2-5 所示的“阶段 1”，用户需要双击屏幕查看“阶段 2”（图 2-6）的效果，但是“阶段 2”部分文字超出屏幕范围，阅读变得非常不便，所以引入 Viewport，对 layout viewport 的尺寸进行控制，以达到“阶段 3”（图 2-7）的效果。

《Viewport 双城记》中提到了 visual viewport 和 layout viewport 概念，layout viewport 是可以通过 meta viewport 属性控制的，visual viewport 是指屏幕上可视的区域；图 2-5 中 visual viewport 和 layout viewport 重合，“阶段 2”中 layout viewport 包含了 visual viewport。


图 2-5　阶段 1：未设置 Viewport

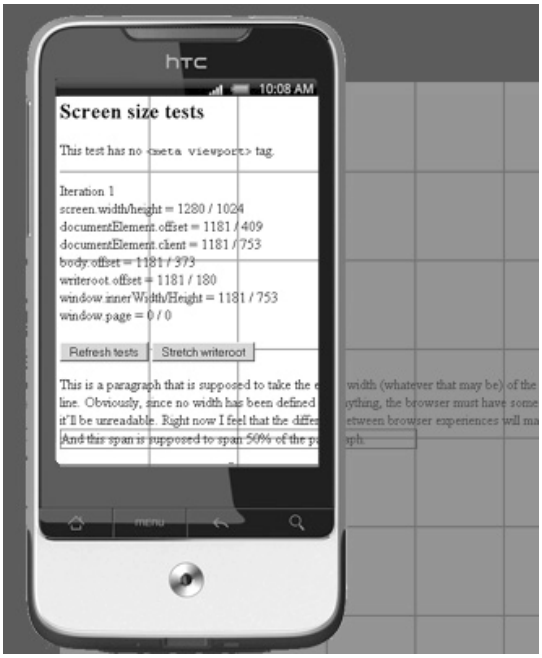



图 2-6 阶段 2：视图放大后

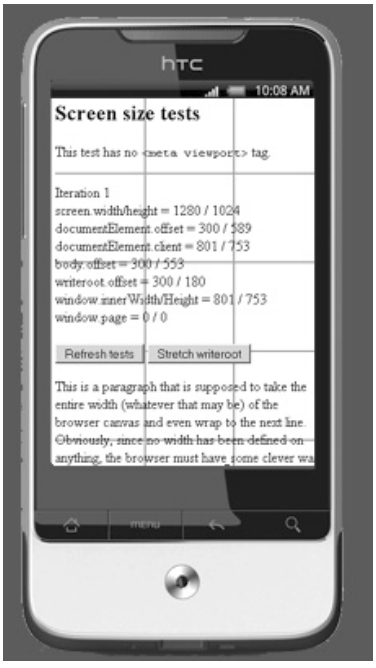



图 2-7 阶段 3：设置 Viewport 后

### 2.2.3 touch-icon

这件事情如果考虑全平台兼容则会比较复杂，比如像这篇文章 *Everything you always wanted to know about touch icons*[12]。apple-touch-icon 到底该怎么写，先给一个结论，以下的结论是基于本书的 MGBS。

兼容 iOS 4.2+和 Android 2.1+的通用写法：

```
<link rel="apple-touch-icon-precomposed" href="http://your.touch.
icon/path" />
```

rel="apple-touch-icon-precomposed"，告诉浏览器不要给 icon 添加额外的效果，另一方面考虑到兼容 Android 低版本（1.5～2.1）。

touch-icon 的尺寸建议使用 144×144，兼容 iPhone、iPad retina 和非 retina 版本[13]，以及 Android 的绝大部分设备。这条规则参考了 m.taobao.com 的 114×114（笔者写此章节时，仍为 114，iPad 3+下显示会被拉伸），可以做得更进一步，使用 144×144，缺点是尺寸略大。如果不这么做意味着：

1. 要兼容 iPad 1/2、iPhone 4+、iPad 3+ 就要在每个页面多写 3 条 link 来控制。
2. Android 屏宽 640+这类设备的兼容还要更复杂，需要 media query 触发。
3. 这个 icon 应该控制在 8KB 以内，不一定放置于根目录，这样就需要在每个页面都使用上面的 link 申明，这是一种更加灵活的方式。

由于 touch-icon 一旦启用将会影响很广，一些细节在此章节完成时仍有待验证。

1. 是否可以通过 HTTP Cache 指令控制更新？
2. 同样一个 touch-icon，在 Android Chrome 下加载一次不再请求，系统自带浏览器每次刷新均请求，可见浏览器在这个特性的实现上是有差异的。
3. iOS root search 问题。
4. Android 对 link 属性 sizes 的支持。
5. Android UC 不支持 rel="apple-touch-icon" ？

---

[12] http://mathiasbynens.be/notes/touch-icons

[13] https://developer.apple.com/library/ios/documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ ConfiguringWebApplications.html

6. Android QQ 浏览器还未验证。

### 2.2.4 其他

**apple-mobile-web-app-capable**

- 从桌面 icon 启动 iOS Safari 是否进入全屏状态。
- 取值：yes | no。
- 判断全屏状态可使用 window.navigator.standalone。

**apple-mobile-web-app-status-bar-style**

- iOS Safari 在启动全屏状态下的状态栏样式。
- 取值：default | black | black-translucent。
- apple-mobile-web-app-capable 必须设为 yes，此项设置才能生效。

**format-detection**

- iOS 设备上禁止将数字识别为可点击的 tel link。

## 2.3 触屏事件

### 2.3.1 触屏事件一览

Android、iOS、Windows Phone（WP）中的触屏事件是基本动作及其组合的结果。Android 基本事件[14]包括触击（Touch）、长按（Long Press）、划动（Swipe）、捏（Pinch），组合事件包括长按并拖动（Long Press Drag）、双击（Double Touch）、双击并拖曳（Double Touch Drag）。

iOS 触屏事件[15]和 Android 大同小异，只是在基本事件的命名和组合事件的定义上稍有差异，表 2-1 显示了 iOS 中几种事件的命名和定义。

[14] http://developer.android.com/design/patterns/gestures.html

[15] https://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/GestureRecognizer_basics/ GestureRecognizer_basics.html#//apple_ref/doc/uid/TP40009541-CH2-SW4

表 2-1　iOS 常见触屏事件

| 事件 | 说明 |
| --- | --- |
| Tapping | 拍击，类似 Android Touch |
| Pinching in and out | 常用于缩放视图 |
| Panning or dragging | 拖曳 |
| Swiping | 划动 |
| Rotating | 旋转 |
| Long Press | 长按 |

Windows Phone 的触屏事件和 Android、iOS 相比命名稍有不同，不再累述。

Hammer.js 是目前使用较广泛的一个事件库，很好地处理了不同移动平台间的触屏事件差异，并对 PC 和 Mobile 的事件做了兼容处理，下面的列表显示了 Hammer 中支持的事件。

```
Hold
Tap
Doubletap
Drag, Dragstart, Dragend, Dragup, Dragdown, Dragleft, Dragright
Dwipe, Swipeup, Swipedown, Swipeleft, Swiperight
Transform, Transformstart, Transformend
Rotate
Pinch, Pinchin, Pinchout
Touch (gesture detection starts)
Release (gesture detection ends)
```

## 2.3.2　通用触屏事件

“触屏事件”[16]这节是笔者最初为 Kissy 的触屏事件[17]所整理，从“触屏事件一览”可以看到通用的触屏事件至少包括：

- 滑动（Swipe），方向上下左右。
- 捏（Pinch），常用于放大（Zoom in）缩小（Zoom out）视图。

[16] 本节讨论基于 https://github.com/kissyteam/kissy-mobile/issues/3。

[17] http://docs.kissyui.com/docs/html/api/core/event/#id2

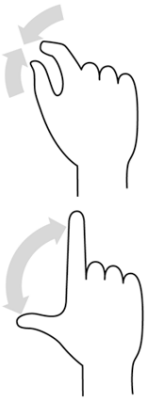
- 旋转（Rotate），常用于旋转视图。
- 拖曳（Drag）。
- 长按（Hold），Android 和 iOS 都叫 Long Press，感觉叫 Hold 更形象。点击稍复杂：
  - Android 称为 Touch，双击是 Double Touch。
  - iOS、WP 和 Hammer 称为 Tap，Hammer 可参考其源码注释。
  - Touch 这个事件本身分为 Touchstart/end/move/cancel。
  - 少数服从多数，就叫 Tap。

表 2-2 给出的是触屏事件中的元事件，组合事件如：Hold+Swipe 在 Android 上用于移动桌面图片。由于组合事件众多，不一一列出。

表 2-2 通用触屏事件

| 事件类别 | 事件描述 | 简称 | 别称 | 动作定义 |
| --- | --- | --- | --- | --- |
| Tap | 移动平台默认浏览器的点击事件有 300ms+的延时，通常使用 Touch 事件模拟，为区别点击称为拍击：<br>`Tap` 拍击<br>`Doubletap` 双击<br>`Hold` 长按<br>`Tapn n`（2,3..）指拍击 | 拍击 | Android：Touch<br>Hold 称呼较多，如下。<br>Android/ios：Long Press<br>Wp：Tap and Hold<br>也有称为 Press |  |
| Swipe | 按方向细分为：<br>`Swipe` 单指滑动<br>`Swipeleft` 单指向左滑动<br>`Swiperight` 单指向右滑动<br>`Swipeup` 单指向上滑动<br>`Swipedown` 单指向下滑动<br>`Swipen` n（2,3..）指滑动 | 滑动 | Wp：Flick |  |

续表

| 事件类别 | 事件描述 | 简称 | 别称 | 动作定义 |
| --- | --- | --- | --- | --- |
| drag | `Drag` 拖曳<br>`Dragstart` 拖曳开始<br>`Dragend` 拖曳结束<br>`Dragup` 向上拖曳<br>`Dragdown` 向下拖曳<br>`Dragleft` 向左拖曳<br>`Dragright` 向右拖曳 | 拖曳 | iOS/WP：Pan |  |
| Pinch | 常用于放大（Zoom in）/缩小（Zoomout）视图：<br>`Pinchin` 双指捏合<br>`Pinchout` 双指展开<br>`Squeeze` 五指捏合<br>`Splay` 五指展开 | 捏 | Android：Pinch open/close<br>Pinchout 也有称为 Spread |  |
| Rotate | 常用于旋转视图：<br>`Rotatecw` 顺时针旋转<br>`Rotateccw` 逆时针旋转 | 旋转 | |  |
| Shake | 常用于游戏中控制方向，细分为：<br>`Shake` 移动设备<br>`Shakeup` 向上移动设备<br>`Shakedown` 向下移动设备<br>`Shakeleft` 向左移动设备<br>`Shakeright` 向右移动设备 | 重力感应 | |  |

续表

| 事件类别 | 事件描述 | 简称 | 别称 | 动作定义 |
| --- | --- | --- | --- | --- |
| | `Shakeforward` 向前移动设备<br>`Shakeback` 向后移动设备<br>`Shakeleftright` 左右移动设备<br>`Shakeforwardback` 前后移动设备<br>`Shakeupdown` 上下移动设备 | | | |

## 2.4 调试

### 2.4.1 远程调试

本节所指的调试泛指开发过程中对页面进行的一切 HTML、CSS、JavaScript 的监控、调优、单点调试等行为。

PC Web 的调试在经历了 IE 低版本的痛苦之后，迎来了像 DebugBar[18]这类的调试工具，IE 8+自带的调试工具勉强能用但也不尽如人意；Firefox 的 Firebug 和 WebKit 的 Web Inspector[19]将调试推向了一个高度。Mobile Web 从诞生的那一天起由于屏幕尺寸的限制，调试就是一个几乎不可能完成的任务[20]。Mobile Web 的调试只能另辟蹊径，基本的思路就是同步 PC 和 Mobile 浏览器环境信息，使用 PC 的大屏完成对只拥有小屏的移动设备的调试工作。得益于 WebKit 启动 Remote Debugging 计划以及之后 Remote Debugging Protocal v1.0 的发布[21]，iOS 和 Android 纷纷跟进并发布了各自的远程调试工具。现在的远程调试比起 3 年前笔者刚进入移动领域开发时已经方便了不少。

---

[18] DebugBar 是 IE 插件，配合 Companion.js，功能接近 Firebug。

[19] http://trac.webkit.org/wiki/WebInspector。Web Inspector 是 WebKit 自带的调试工具。Safari 和 Chrome 中的开发者工具均是基于 Web Inspector 的。

[20] Mission Impossible 电影《碟中谍》系列。

[21] https://www.webkit.org/blog/1875/announcing-remote-debugging-protocol-v1-0/

Mobile Web 上的主要的调试场景为两类：系统自带浏览器（iOS Safari、Android Chrome）是第 1 类、其他浏览器和 WebView 为第 2 类。

### 2.4.1.1　Mobile Emulation

页面在开始远程调试之前，应该在 Mobile Emulation 上调通，这对开发效率的提升很有帮助。以 PC Chrome Mobile Emulation（Chrome 34.0.1756.0 canary）为例（图 2-8），打开开发者工具，在红色线框加亮部分可以选择常见的移动设备，单击“Emulate”选项卡进入模拟；此时“Screen”、“UserAgent”等已经设置好，可以根据需要改写这些属性。Chrome 的最新稳定版本（lastest stable）界面类似。

PC Safari 可以通过菜单“开发→用户代理”设置模拟设备，再开启开发者工具，如图 2-9 所示。

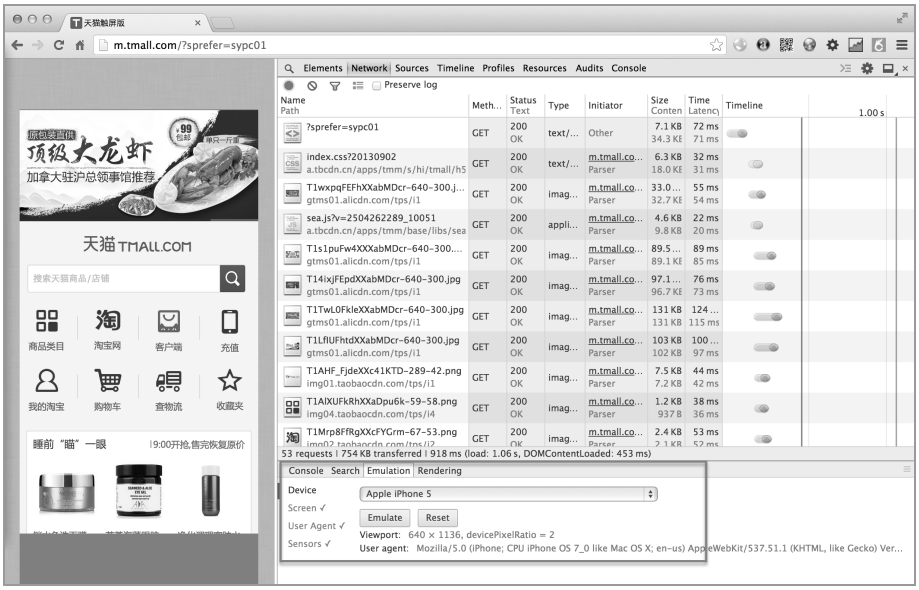



图 2-8　PC Chrome Mobile Emulation（Chrome 34.0.1756.0 canary）

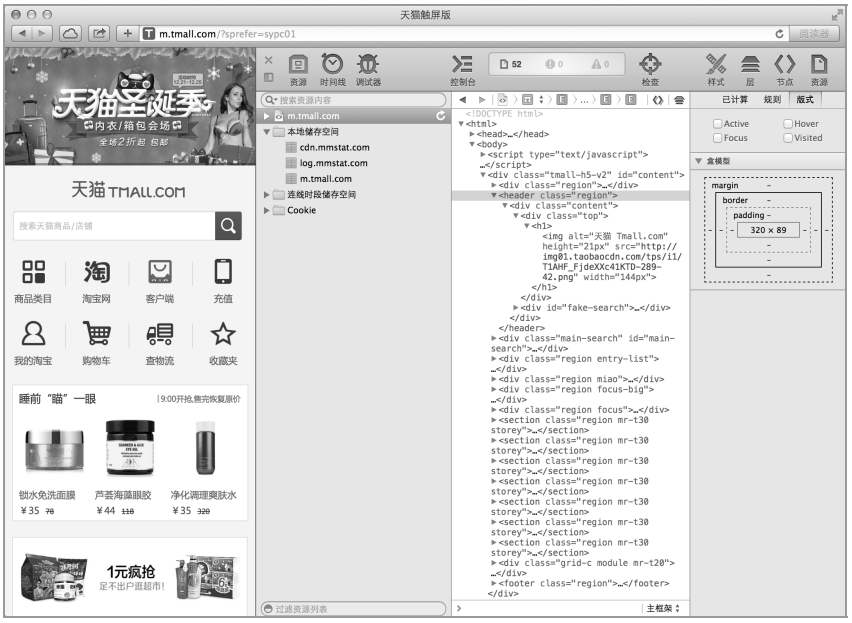



图 2-9 PC Safari Mobile Emulation

### 2.4.1.2 iOS 远程调试

iOS 6+只要几步简单的操作，便可以使用 OS X（iMac、Macbook 等）Safari 6+调试 iOS Safari 或包含有 WebView 的 App（如使用 PhoneGap 开发的 App）。

**步骤 1**：开启 OS X Safari 的“开发”菜单，如图 2-10 所示。操作顺序：Safari→Safari（菜单）→偏好设置→高级。


图 2-10 步骤 1：开启 OS X Safari 的“开发”菜单

**步骤 2**：开启 iOS Safari 的 Web 检查器，如图 2-11 所示。操作顺序：设置→Safari→高级。

**步骤 3**：OS X Safari 连接 iOS Safari 操作顺序：OS X Safari→开发（菜单）→设备名称（图 2-12 中的“徐凯的 iPhone”）。

**步骤 4**：OS X Safari 会弹出一个熟悉的 Web Inspector 界面，之后的操作和在 PC 上的调试类似，如图 2-13 所示。


图 2-11　步骤 2：开启 iOS Safari Web 检查器


图 2-12　步骤 3：OS X Safari 连接 iOS Safari


图 2-13　步骤 4：连接后的 Web Inspector 界面

以上描述了 iOS Safari 中的远程调试过程，如果需要在 iOS WebView 中进行远程调试[22]，需要设置 UIWebView 为调试模式[23]，之后的远程调试过程和 iOS Safari 完全一样。

```
- ( BOOL ) application: ( UIApplication * ) application
didFinishLaunchingWithOptions: (NSDictionary *) launchOptions {
  // ......
  [NSClassFromString (@"WebView") _enableRemoteInspector];
  // ......
}
```

### 2.4.1.3 Android 远程调试

Android Chrome 在远程调试上持续改进，到笔者撰写该节时 Paul Irish 在 html5rocks 上发表了 *Chrome DevTools for Mobile: Screencast and Emulation*[24]，Chrome DevTools 已经发布了屏幕映射（screencast）功能，这让远程调试变得更加方便了，如图 2-14 所示。

图 2-14 屏幕映射（screencast）

[22] http://twilight.btlogs.com/remote-debug-iphone-mobile-safari-for-ios-using-webkit-inspector-with-private-api/
这篇文章提供了一个可用于远程调试的 iOS App 的源码。
http://twilight.btlogs.com/files/2012/03/ios-remote-debug.zip

[23] http://atnan.com/blog/2011/11/17/enabling-remote-debugging-via-private-apis-in-mobile-safari/

[24] http://www.html5rocks.com/en/tutorials/developertools/mobile/

Android Developer 上的 *Remote Debugging Chrome on Android*[25]详细讲解了如何使用 PC Chrome 连接、远程调试、屏幕映射等。

**步骤 1**：安装 Android Chrome 31+，其中 32+才有屏幕映射功能。

**步骤 2**：使用 USB 线连接移动设备和 PC（Windows 用户需要安装合适的 USB 驱动），并且需要开启 Android 的“USB debugging”，如图 2-15 所示。

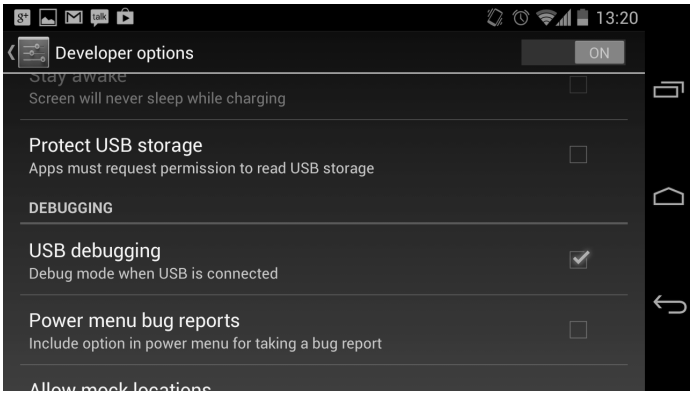



图 2-15　步骤 2：开启“USB debugging”

**步骤 3**：安装 ADB Extension[26]，如图 2-16 所示。PC Chrome 32+自带 Native USB Debugging 功能，可以跳过该步骤，安装后需要手工启动 ADB。

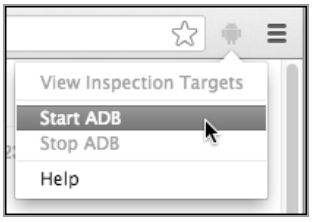



图 2-16　步骤 3：启动 ADB

**步骤 4**：PC Chrome 31+。使用 about:inspect 开启远程调试的控制台，单击“Inspect”开始远程调试，如图 2-17 所示。

---

[25] https://developers.google.com/chrome-developer-tools/docs/remote-debugging

[26] https://chrome.google.com/webstore/detail/dpngiggdglpdnjdoaefidgiigpemgage


图 2-17　步骤 4：远程调试界面（Android Chrome 32+、PC Chrome 31+、ADB Extension）

尽管 Chrome 团队为 Android 的远程调试做了持续有效的努力，但是 WebView 页面的远程调试问题直到 Android 4.4 才有所突破[27]（见代码 2-4），需要对 WebView 的调用做如下操作，如图 2-18 所示，之后的远程调试步骤和上面介绍的相似。这段代码可以从此处获得，包含源代码和一个 APK 安装包[28]。

**代码 2–4　Android 4.4+开启远程调试的代码**

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
    WebView.setWebContentsDebuggingEnabled(true);
}
```

[27] https://developers.google.com/chrome-developer-tools/docs/remote-debugging#debugging-Webs

[28] 源代码：http://luics.github.io/cew/Bridge.zip

安装包：http://luics.github.io/cew/Bridge.apk

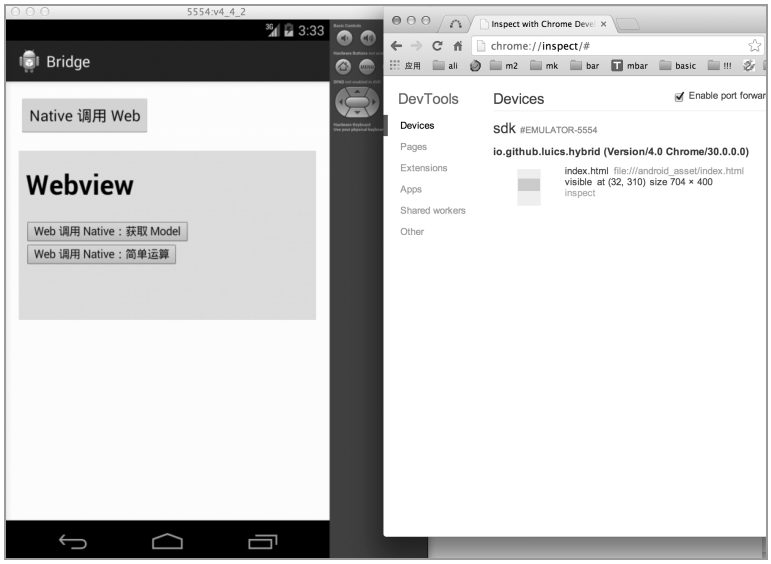



图 2-18　Android 4.4.2（API 19）开启 WebView 远程调试

### 2.4.1.4　weinre

至此，iOS 和 Android 都有了各自的原生远程调试方式，可惜 Android 4.4 版本的固件直到 2014 年 2 月仍然很少见，面对庞大的远程调试需求必须要有替代的办法，iOS 的情况也仅仅是略好于 Android。这也正是需要“weinre”和“HTTP 代理服务器”这两节的原因。

weinre 最初是 PhoneGap[29]的子项目，Adobe 收购了 PhoneGap 的母公司后推出的 Edge Inspect[30]（最初名为 Shadow）也使用了 weinre 的远程调试技术。截止笔者本书初稿（2013 年 12 月），weinre 仍然是 iOS Safari 6+和 Android Chrome 31+自带调试工具之外最强大的调试工具，也是迄今为止最通用（跨 Android 和 iOS）的调试工具，可以方便地调试各种独立浏览器和 WebView。weinre 开启了远程调试的大门，也促使之后 WebKit 宣布开发自己的远程调试工具，并拟立了远程调试规范，才有了今天 iOS Safari 和 Android Chrome 上不断改进的远程调试体验。如图 2-19 所示为 weinre 远程调试示例。

---

29　http://phonegap.com/PhoneGap 是主流的 Hybrid App 框架。

30　http://html.adobe.com/edge/inspect/

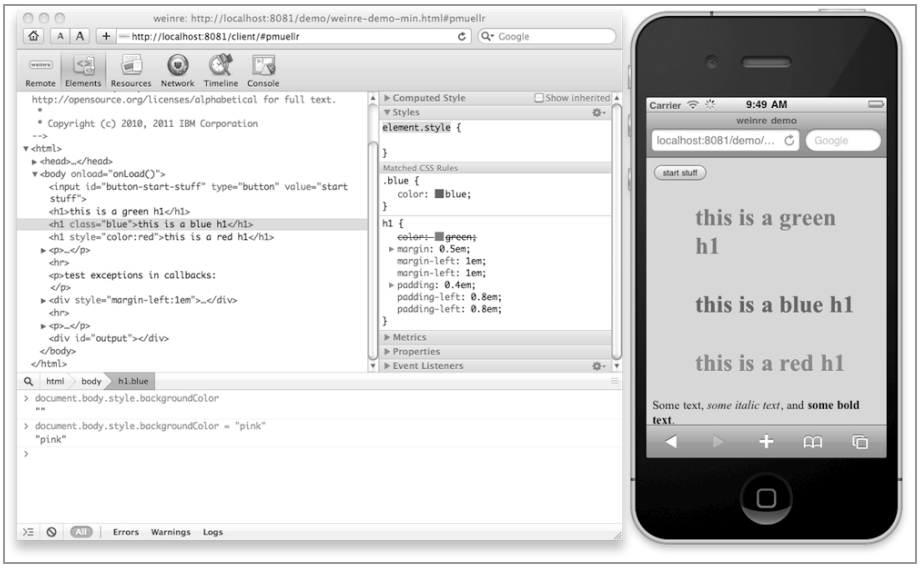



图 2-19 weinre 远程调试示例

在介绍 weinre 的具体使用方式之前，先介绍 3 个概念。

1. Debug Server，使用 weinre 命令启动的一个 http server，Debug Client 和 Debug Target 均会与 Debug Server 通信。
2. Debug Client，提供了“Elements”、“Console”这两个基本的调试工具，Debug Client 复用了 WebKit Inspector 的界面。
3. Debug Target，需要远程调试的页面，由于 weinre 最初用于 PhoneGap 框架开发的 Hybrid App 的远程调试，所以还包括运行在 WebView 中的那些页面。

以在 Client 的“Console”面板上执行一段 js 脚本为例：

```
document.body.style.backgroundColor = "pink";
```

Client 与 Target 均与 Server 建立了长连接（最新版本 weinre 使用 WebSocket）；Client 将脚本封装为一个命令传至 Server，Server 再通过长连接通知到 Target，最终 js 脚本在 Target 上执行，执行的结果由 Target 传至 Server 后通过长连接再通知到 Client，最终 Client 的界面上显示“pink”。

Weinre 时序图如图 2-20 所示。

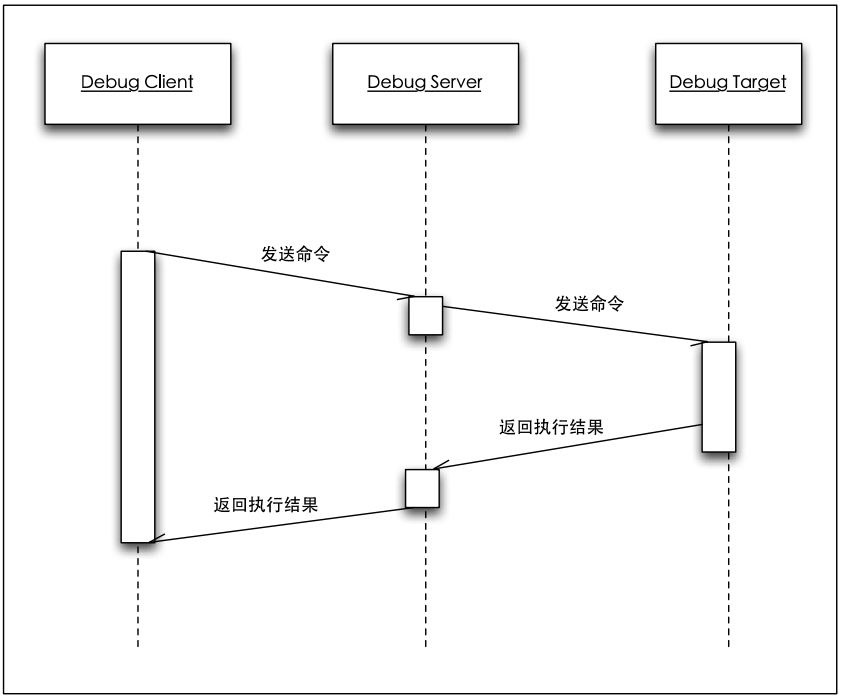



图 2-20　weinre 时序图

下面简单介绍 weinre 的使用方式。

1. 安装。现在通过 npm[31]安装非常方便，jar 包的方式不再推荐了。
```
npm install -g weinre
```
2. 启动 Debug Server。“--httpPort 8080”指定 Server 监听 8080 端口，“--boundHost -all-”指定 Server 监听所有地址，默认只监听 localhost。
```
weinre --httpPort 8080 --boundHost -all-
```
3. Debug Target 添加如下脚本。“{ip}”替换为 Debug Server 的 ip，这样便于 Mobile Web Page 访问。
```
<script src="http://{ip}:8080/target/target-script-min.js">
</script>
```
4. 访问 Debug Client。会看到类似图 2-21 的界面。
```
http://localhost:8080
```

[31] npm 是 node.js 的包管理工具，安装 node.js（http://node.js.org）后就可以使用 npm 命令。

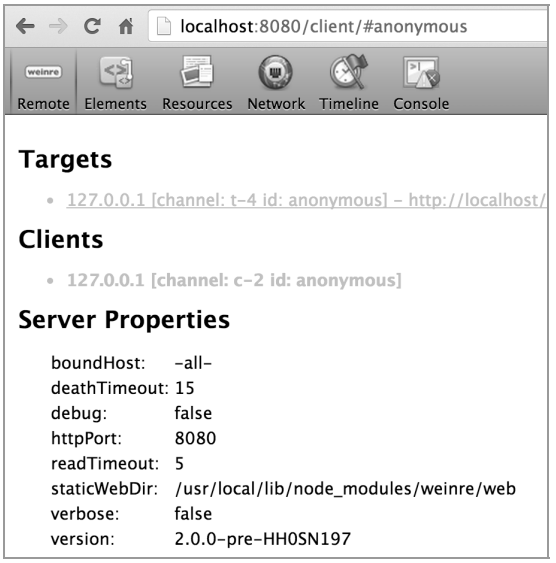



图 2-21 weinre 的 Debug Client

步骤 3 需要在调试的页面中硬编码 weinre 的脚本，有时候不方便改写页面代码时，还可以将如下代码保存为书签，在需要的时候打开该书签便可实现 weinre 脚本注入，这种技术也被称为 bookmarklet。下面代码中的 localhost 请替换为 Debug Server IP，便于在移动端使用。

```
javascript:(function(e){e.setAttribute("src","http://localhost:
8080/target/target- script-min.js#anonymous");document.getElementsByTagName
("body")[0].appendChild(e);})(document.createElement("script"));void
(0);
```

weinre 在很大程度上满足了远程调试的需求，但由于 weinre 技术实现上的限制，无法实现网络（Network）、资源（Resources）、性能监控（Profile）、时间轴（Timeline）等调试功能。

### 2.4.1.5 HTTP 代理服务器

HTTP 代理服务器（Proxy）可以弥补 weinre 在网络调试上的先天不足，将移动设备的 WIFI HTTP 代理服务器地址设置为 PC HTTP 代理服务器（Proxy Server）的地址即可。最常用的 Proxy Server 是 Fiddler（OS X 系统可以使用 Charles）。

以 Fiddler 为例进行操作。

**步骤 1**：开启 Fiddler 远程连接，需要勾选“Allow remote computers to connect”复选框，如图 2-22 所示。

**步骤 2**：开启 iOS 或 Android 的 WIFI 代理。

- iOS 的 WIFI 代理设置为：设置→无线局域网→某热点的属性→HTTP 代理→手动，如图 2-23 所示。
- Android 的 WIFI 设置为：系统设置→WLAN→某热点的属性→代理设置→手动。

Android 中自带浏览器和 Chrome 的网络流量默认会过 WIFI 代理，但是其他应用则不会，需要安装全局代理工具[32]才能生效。

**步骤 3**：使用 Fiddler，这一步和在 PC 上使用 Fiddler 无差异，如图 2-24 所示。

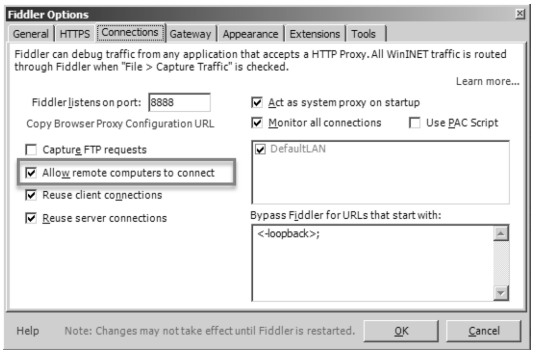



图 2-22　步骤 1：开启 Fiddler 远程连接


图 2-23　步骤 2：iOS WIFI 代理

[32] Android 的全局代理工具有 ProxyDroid 等，需要 root 权限。

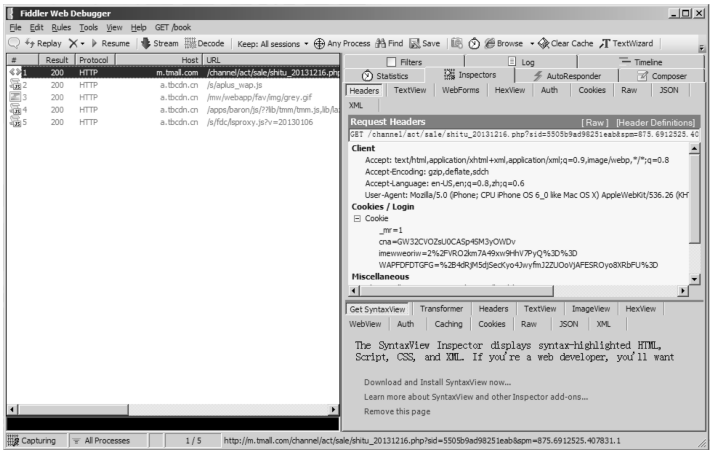



图 2-24 步骤 3：使用 Fiddler

## 2.4.2 设备调试

### 2.4.2.1 设备模拟器

OS X 设备安装了最新版本的 xCode 之后，可以启动 iPhone 和 iPad 模拟器。安装最新版本 Eclispe 和 ADT[33]插件，或安装最新版本的 Android Studio[34]，在启动的 Android Virtual Device Manager 中可以启动模拟器，如图 2-25 所示。

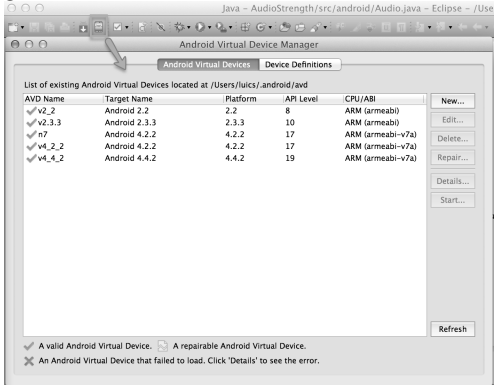



图 2-25 Android Virtual Device Manager

[33] ADT（Android Development Toolkit）http://developer.android.com/tools/sdk/eclipse-adt.html

[34] http://developer.android.com/sdk/installing/studio.html

#### 2.4.2.2 远程设备

硬件设备的购置通常是一笔不小的费用，并且设备的利用率也较低。远程设备正是为了解决这个问题而产生的。

远程设备的设计思路通常如下。

1. 开发工具将设备屏幕实时映射到客户端。
2. 客户端接受用户点击、划动、拖曳等动作，并将动作映射到设备，从而完成用户“操作”设备功能。
3. 客户端的动作扩展，比如支持摇一摇、音频输入、视频输入等功能。
4. 开发功能扩展，比如支持 App 测试和认证、客户端日志输出等功能。目前国内比较成熟的远程设备平台有：
   - 移动云测试中心（MTC，Mobile Testing Center）[35]，百度开放云平台下的一款产品，目前已经成长为包含云测试、云调试、录制回放、安全扫描、云众测等功能的综合性远程设备平台。笔者曾有幸参与 MTC Alpha 版本的开发工作中。
   - 云测（Testin）[36]，国内较早成立的远程设备平台，提供了大量开发功能扩展。

## 2.5 兼容性

尽管 Android 和 iOS 平台的大部分独立浏览器以及 WebView 都使用 WebKit 内核，尽管移动端不存在像 IE 6/7 那样古老的浏览器，但是移动端仍然存在较多兼容性问题，表现为如下几方面。

### 2.5.1 OS 版本碎片化

过于分裂的 Android 版本和轻度分裂的 iOS 版本都导致其自带浏览器 WebKit 内核版本的差异，从而导致可用性上存在兼容性问题，这类兼容性问题通过类似

[35] http://mtc.baidu.com/

[36] http://www.testin.cn/

caniuse.com[37]的工具通常都能查阅到，相对容易预测和处理。如图 2-26 所示为 iOS 和 Android 版本变化趋势。

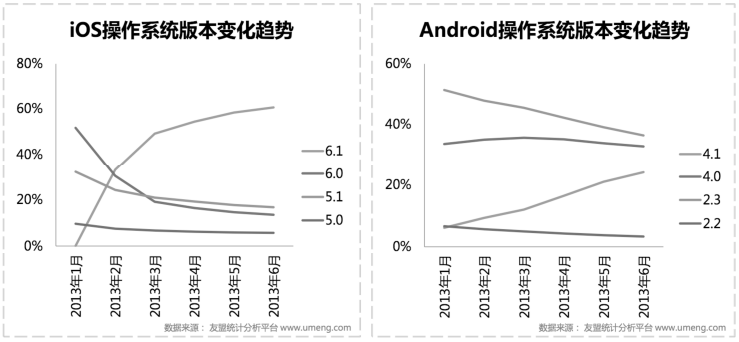
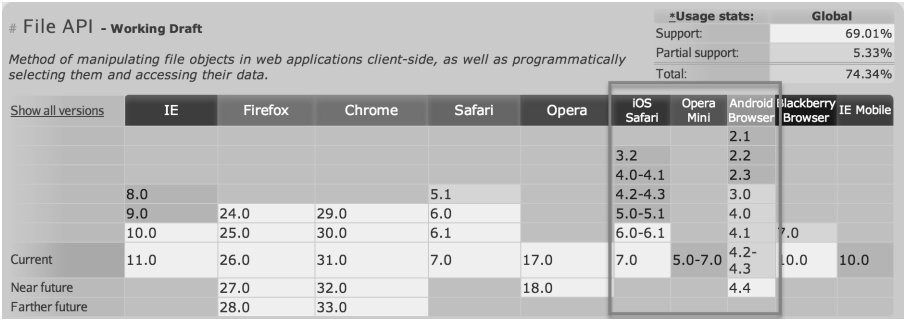



图 2-26 iOS 和 Android 版本变化趋势[38]

File API 的兼容性体现出 OS 版本碎片化的现状，尤其是在 Android 2.3 仍高达近 20%的情况下。如图 2-27 所示为 caniuse.com File API 查询结果。



# File API - Working Draft

*Method of manipulating file objects in web applications client-side, as well as programmatically selecting them and accessing their data.*

| *Usage stats: | Global |
|---|---|
| Support: | 69.01% |
| Partial support: | 5.33% |
| Total: | 74.34% |

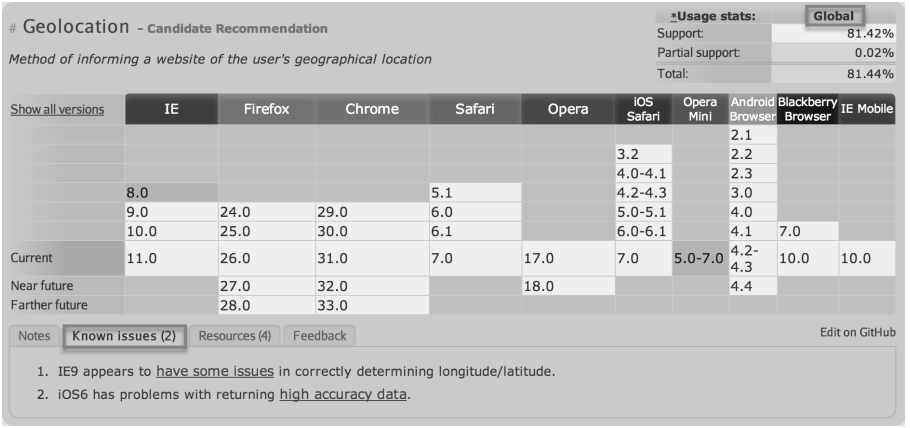
| Show all versions | IE | Firefox | Chrome | Safari | Opera | iOS Safari | Opera Mini | Android Browser | Blackberry Browser | IE Mobile |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 2.1 | | |
| | | | | | | 3.2 | | 2.2 | | |
| | | | | | | 4.0-4.1 | | 2.3 | | |
| | 8.0 | | | 5.1 | | 4.2-4.3 | | 3.0 | | |
| | 9.0 | 24.0 | 29.0 | 6.0 | | 5.0-5.1 | | 4.0 | | |
| | 10.0 | 25.0 | 30.0 | 6.1 | | 6.0-6.1 | | 4.1 | 7.0 | |
| Current | 11.0 | 26.0 | 31.0 | 7.0 | 17.0 | 7.0 | 5.0-7.0 | 4.2-4.3 | 10.0 | 10.0 |
| Near future | | 27.0 | 32.0 | | 18.0 | | | 4.4 | | |
| Farther future | | 28.0 | 33.0 | | | | | | | |

图 2-27 caniuse.com File API 查询结果

再以 Geolocation 为例，图 2-28 主体区域显示了各个平台上的支持情况，包括 PC 和 Mobile 浏览器，从图中右上角“Global”可以看到 Geolocation 在全平台的支持情况，从左下角可以看到已知问题和资源（W3C 规范、Demo 等）。

[37] http://caniuse.com/

[38] 摘自《友盟 2013 上半年报告》。



# Geolocation - Candidate Recommendation

*Method of informing a website of the user's geographical location*

*Usage stats: Global

Support: 81.42%
Partial support: 0.02%
Total: 81.44%

| Show all versions | IE | Firefox | Chrome | Safari | Opera | iOS Safari | Opera Mini | Android Browser | Blackberry Browser | IE Mobile |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 2.1 | | |
| | | | | | | 3.2 | | 2.2 | | |
| | | | | | | 4.0-4.1 | | 2.3 | | |
| | 8.0 | | | 5.1 | | 4.2-4.3 | | 3.0 | | |
| | 9.0 | 24.0 | 29.0 | 6.0 | | 5.0-5.1 | | 4.0 | | |
| | 10.0 | 25.0 | 30.0 | 6.1 | | 6.0-6.1 | | 4.1 | 7.0 | |
| Current | 11.0 | 26.0 | 31.0 | 7.0 | 17.0 | 7.0 | 5.0-7.0 | 4.2-4.3 | 10.0 | 10.0 |
| Near future | | 27.0 | 32.0 | | 18.0 | | | 4.4 | | |
| Farther future | | 28.0 | 33.0 | | | | | | | |

Notes　Known issues (2)　Resources (4)　Feedback　Edit on GitHub

1. IE9 appears to have some issues in correctly determining longitude/latitude.
2. iOS6 has problems with returning high accuracy data.

图 2-28　caniuse.com Geolocation 查询结果

## 2.5.2　国内的特殊情况

国内 Android 浏览器第一阵营的系统自带浏览器、QQ 浏览器、UC 浏览器[39]，虽然都使用 WebKit，但是 QQ 浏览器和 UC 浏览器又有不同程度的定制，并且相关的兼容性问题通常存在于特定浏览器的特定版本，没有一个方便查阅的资料库，处理起来非常棘手。

## 2.5.3　WebView

无论 Android 还是 iOS，WebView 和系统自带浏览器使用了相同的内核，按理二者兼容性应该很好，但是 WebView 提供了大量 API，可以由开发者控制某些特性的开启或关闭，这会导致部分特性在自带浏览器下可用而在 WebView 下不可用。这个问题在对 WebView 不可控的场景尤为棘手，此时只能依靠特性检测来做兼容处理，可以考虑下文提到的 Modernizr。

## 2.5.4　更多工具

前面提到了 caniuse，本节再罗列一些常用的兼容性工具。

[39] 浏览器分布情况请参见本书“MGBS”章节。

html5test.com[40]可以检测当前访问的浏览器对 HTML5 特性的支持程度，提供非常详细的检测结果，同时还支持与其他浏览器检测结果进行对比。如图 2-29 所示为 html5test 检测结果。

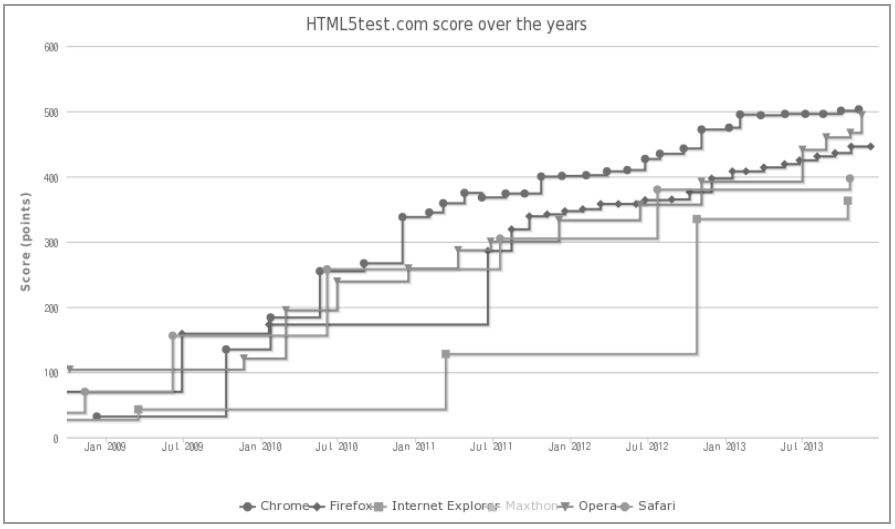



图 2-29 html5test 检测结果

mobilehtml5.com[41]提供了一个兼容性矩阵，可以更全局直观地查阅兼容性；html5readiness.com[42]以时间纬度展示所有浏览器 HTML5 特性的可用性，数据来自 caniuse.com；此外 Chrome[43]、Firefox[44]、IE[45]也提供了各自对 HTML5 的支持情况；W3C 也在自家的 HTML[46]/CSS[47]Validation 中添加了对 HTML5、CSS3 的支持。

再提一下特性检测库 Modernizr[48]（见图 2-30），Modernizr 通过对特性检测并提供对

40 http://html5test.com/

41 http://mobilehtml5.org/

42 http://html5readiness.com/

43 http://dev.chromium.org/developers/web-platform-status

44 https://developer.mozilla.org/cn/html/html5

45 http://msdn.microsoft.com/en-us/ie/hh272905.aspx

46 http://validator.w3.org/

47 http://jigsaw.w3.org/css-validator/

48 http://modernizr.com/

低版本浏览器的优雅降级处理，让渐进增强的代码更容易维护，这在 HTML5 兼容性问题还广泛存在的今天还是很有意义的。此外 Modernizr 还提供了一个 Polyfill[49]加载机制，可以让低版本浏览器支持某些高级特性，比如在不支持 Websocket 的浏览器中加载一个 Websocket polyfill 可以让该浏览器支持 Websocket 特性。更多 Polyfill 的讨论可以参考 Paul Irish 写的这篇文章 *HTML5 Cross Browser Polyfills*[50]。

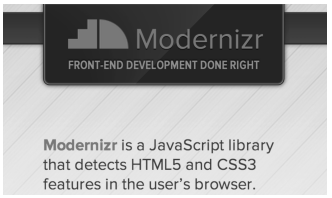



图 2-30　Modernizr

## 2.6　文档

API 文档可以根据需要选择最适合自己的一款，笔者推荐 MDN（Mozilla Developer Network）[51]。2013 年 12 月 MDN 进行了一次改版，改版后的 MDN 信息结构更加合理，UI 更简洁和扁平化，如图 2-31 所示。

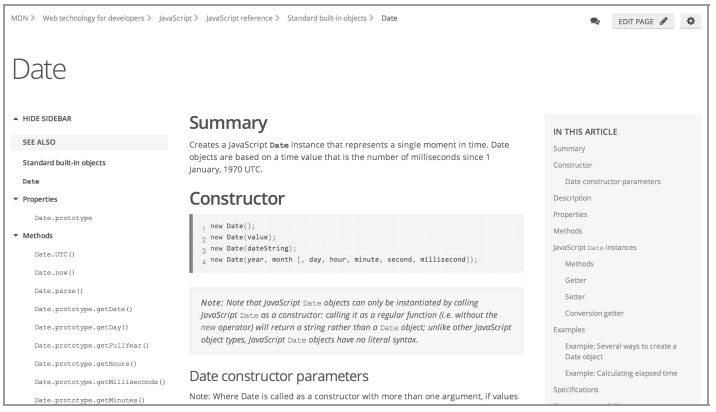



图 2-31　改版后的 MDN

---

[49] Polyfill 是一种用于平整墙面裂缝的材料，作为前端术语通常指实现某个特性的兼容处理方案，相近的名词是 shim。

[50] https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills

[51] https://developer.mozilla.org

MDN 几乎涵盖了 Web 所有重要的内容，MDN 的维护者 Mozilla 一直以来也是 W3C 的中坚力量。笔者偶尔逛一次 MDN 时几乎都能有一些意外的惊喜[52]，能有这样一款 API 手册伴随左右，感觉很踏实。

由 Google 支持的 html5rocks.com[53]已经成长为 Web 新技术交流的中心之一，不断有高质量的文章出现。ALA（AListApart）[54]作为 Web 新概念的发源地之一已经存在有 10 余年。移动互联网时代 ALA 同样引领 Web 技术的潮流，2010 年出现的响应式设计（Responsive Web Design）[55]就是出自 ALA。

另外订阅各类“Weekly”，如 HTML5 Weekly[56]、JavaScript Weekly[57]、CSS Weekly[58]、Response Design Weekly[59]、Web Design Weekly[60]，同样是跟进 Mobile Web 技术发展的必要手段。

[52] 比如这套 CSS 小工具集：https://developer.mozilla.org/en-US/docs/Web/CSS/Tools。

[53] http://www.html5rocks.com/

[54] http://alistapart.com/

[55] http://alistapart.com/article/responsive-web-design

[56] http://html5weekly.com/

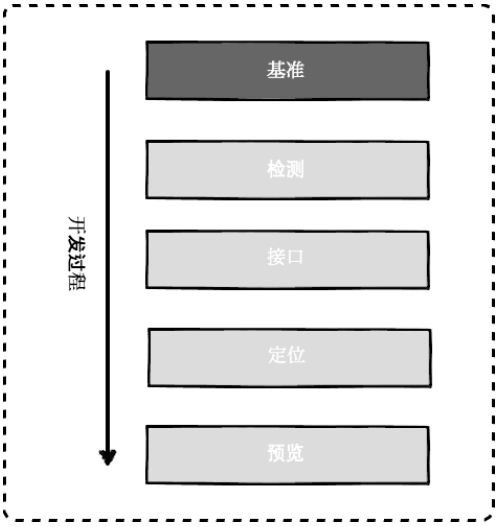
[57] http://javascriptweekly.com/

[58] http://css-weekly.com/

[59] http://responsivedesignweekly.com/

[60] http://web-design-weekly.com/

# 3 基准




由 1.2 节例子进入开发后遇到的第一个问题便是：应该在哪些设备上调试和测试？

这个问题产生的根源是市场上存在众多的设备（操作系统）、不同的屏幕尺寸（分辨率）以及不同的自带或第三方浏览器（浏览器）。如果市场上只有一种移动操作系统且版本时刻最新、只有一种屏幕分辨率、只有一款浏览器且版本时刻最新，那么本章完全就是多余的。可事实并非如此。

## 3.1 GBS

Yahoo 在 2008 年提出了 GBS（Graded Browser Support，分级浏览器支持）来应对日益增长的多浏览兼容问题。尽管 GBS 是在 PC 时代提出的，其设计思路仍然适用于

Mobile 环境。本节是 GBS 的一个简要总结，为便于读者深入理解，笔者翻译了 GBS 原文，详见“附录 GBS”。

引用 GBS 中的一段定义：

> “浏览器测试基准提供一套基准测试的浏览器。它旨在通过测试浏览器组合的最小可能子集，并利用共享的核心浏览器引擎的隐式覆盖率，以有限的测试资源最大化测试覆盖率。为了达到“基准”覆盖率，所有列出的浏览器至少应在一个操作系统中测试。一个特性如果含有特定操作系统下的兼容性问题且在单个操作系统下达到基准覆盖率时，才需要针对多个操作系统进行测试。测试操作系统的选择应该基于使用率和市场趋势。”

可见 GBS 的核心价值在于“以有限的测试资源最大化测试覆盖率”。实践中，GBS 采用的原则是：

> “浏览器测试基准定义了具有验证过的可用的用户体验的一组浏览器。试图为所有测试浏览器提供 A-级（见下文）体验既不经济也不常见。我们建议采用分层的办法对用户体验进行设计、开发和测试，并鼓励每个项目定义最适合于目标用户和测试资源的层级。”

这里提到了分层的概念，GBS 将待测试的浏览器分为 C、A、X 这 3 个级别。

- C 级是支持的基础级别，提供核心的内容和方法。有时被称作核心支持。只通过语义化的 HTML 定义，内容和体验高度可访问，可被装饰或高级功能削弱，具有很好的前向和后向兼容性。样式和行为层不在 C 级考虑范围。
- A 级是最高的支持级别。为了充分利用现代 Web 标准的强大能力，A 级体验包括了高级功能和视觉保真度。
- X 级是为未知的、零散的、很少使用或停止开发的浏览器而准备的。

最后提一下优雅降级和渐进式增强。优雅降级允许较少使用的浏览器提供更少的内容给用户。渐进式增强以内容为核心，允许最多使用的浏览器展现更多内容给用户。虽然概念相近，渐进式增强却是更为健康和有前瞻性的方法，因而渐进式增强才是分级浏览器支持的核心概念。

## 3.2 MGBS

本节我们将讨论如何构造移动版本的 GBS，姑且称之为 MGBS（Mobile Graded Browser Support 移动分级浏览器支持）。

尽管 GBS 是 Netscape 4.x 时代的产物，但是 GBS 的理念本身不会随时间推移而褪色。我们看到 GBS 中对于支持及支持分级的定义，以及渐进式增强在浏览器分级乃至前端技术分级上的论述，都让我们受益匪浅。

今天的移动互联网方兴未艾，情况像极了 2000 年前后的 PC 互联网。

1. 面临的移动浏览器数量庞大，而且还在不断激增，但是占据主导地位的仅有数个而已。
2. 移动互联网仍处于发展初期，操作系统的差异仍然很大，并且呈现出 Android 和 iOS 势均力敌的态势，这使得分级上不得不考虑操作系统因素。
3. PC 时代的屏幕分辨率相对统一，移动的情况则要糟糕得多。

要为移动互联网制定一个合理的分级标准，充满了挑战，以下内容将先从数据源的选择开始，之后依次讨论了操作系统、屏幕分辨率、浏览器的分级，最后总结得出 MGBS。

### 3.2.1 准备

#### 3.2.1.1 数据源

基于统计数据源的覆盖面选择最有参考价值的数据来源，并且对比多个数据源，确保结论的准确可靠。

- 淘宝无线数读（WDM 主要依据）[1]
  - 数据指数，隔天更新，数据来自集团 90%的 App。
  - 数据报告，季报、年度报。
- 友盟（umeng 主要依据）
  - 移动 App 统计市场占有率第一。

[1] 淘宝无线数读：http://wdm.taobao.com/pub2/publicos.htm。

- 2013 年 3 月覆盖应用 10 万+。
- 资源：友盟数据报告[2]、友盟指数[3]。

- 百度移动统计（佐证）[4]
  - 移动 App 统计市场占有率第二，仅次于友盟。
  - 资源：百度移动统计数据报告。
- 其他数据源
  - Android Dashboard[5]，Google 维护的全球 Android 数据（更新频率两周一次），和国内的数据会有差异，但在大趋势上有参考价值。
  - 中国互联网信息中心（CNNIC）[6]，不定期发布移动互联网相关报告，包括：手机浏览器报告、中国网民行为报告等。
  - 百度无线数据报告[7]，数据主要来自百度无线搜索，季报形式发布无线数据。
  - Net Application[8]。

以下几节主要使用 WDM 和友盟的数据。

#### 3.2.1.2 MGBS 中的级别

和 GBS 中根据特性的分级不同，考虑到可操作性，MGBS 中会将浏览器、操作系统、屏幕分辨率划分为 A、B、C 三级。A 级优先级最高，B 级次之，C 级为可选。

### 3.2.2 操作系统分级

#### 3.2.2.1 Android 版本分布

Android 版本分布的数据来自：

---

2 友盟数据报告：http://www.umeng.com/umengdata_reports。

3 友盟指数：http://www.umindex.com/。

4 百度移动统计：http://mtj.baidu.com/web/welcome/industry。

5 Android Dashboard：http://developer.android.com/about/dashboards/index.html。

6 中国互联网信息中心：http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/。

7 百度无线数据报告：http://developer.baidu.com/report。

8 Net Application：http://marketshare.hitslink.com/browser-market-share.aspx?qprid=1。

- 友盟指数→操作系统（截图日期 20140205），如图 3-1 所示。
- 无线数读→无线数据指数→操作系统（截图日期 20140205），如图 3-2 所示。

| | | |
|---|---|---|
| 1 | 4.1.2 | 18.8% |
| 2 | 4.0.4 | 17.8% |
| 3 | 4.1.1 | 11.5% |
| 4 | 4.2.2 | 10.8% |
| 5 | 2.3.6 | 8.5% |
| 6 | 2.3.5 | 7.8% |
| 7 | 4.2.1 | 7.2% |
| 8 | 4.0.3 | 6.4% |
| 9 | 4.3 | 2.4% |
| 10 | 2.3.4 | 2.3% |
| 11 | 2.3.7 | 1.9% |
| 12 | 其他 | 4.7% |

图 3-1 Android 版本分布（友盟指数）

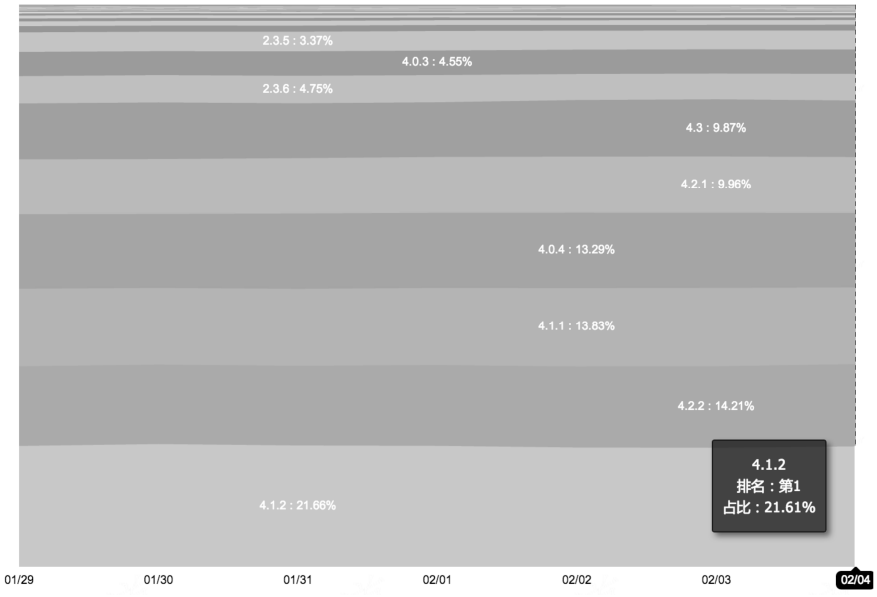



图 3-2 Android 版本分布（WDM）

将以上 Android 操作系统版本划分为：4.4.x、4.3.x（含 4.2.x）[9]、4.1.x、4.0.x、2.3.x、其他。在数据的选择上笔者统计了数值在 0.1%以上的数据，得到图 3-3、图 3-4 和图 3-5。

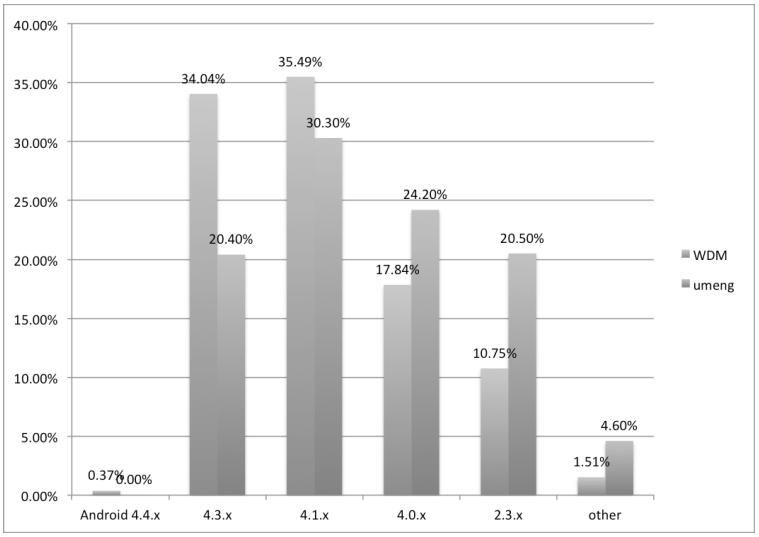



图 3-3　Android 版本分布对比

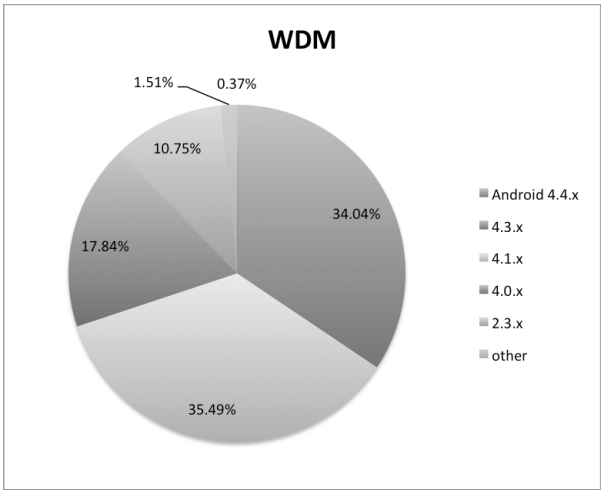



图 3-4　WDM 的 Android 版本分布

[9] 此处参考了 caniuse 中的划分方式 http://caniuse.com/#feat=audio。

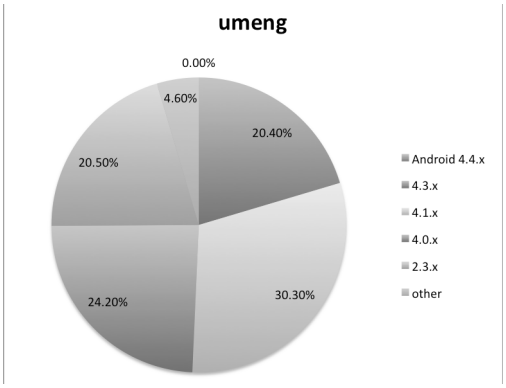



图 3-5 友盟指数的 Android 版本分布

### 3.2.2.2 iOS 版本分布

iOS 版本分布的数据来自：

- 友盟指数→操作系统（截图日期 20140205），如图 3-6 所示。
- 无线数读→无线数据指数→操作系统（截图日期 20140205），如图 3-7 所示。

由于友盟指数在 2014 年 2 月 5 日仍然只提供了 2013 年 12 月的数据，因而图 3-6“iOS 版本分布”中缺少 7.0.5。

| | | |
|---|---|---|
| 1 | 7.0.4 | 40.1% |
| 2 | 7.0.3 | 11.9% |
| 3 | 6.1.3 | 9.7% |
| 4 | 7.0.2 | 7.9% |
| 5 | 5.1.1 | 6.9% |
| 6 | 6.1.2 | 4.1% |
| 7 | 7.0 | 4.1% |
| 8 | 6.0.1 | 3.0% |
| 9 | 6.1 | 3.0% |
| 10 | 6.1.4 | 2.4% |
| 11 | 5.0.1 | 2.4% |
| 12 | 6.0 | 1.0% |
| 13 | 其他 | 3.4% |

图 3-6 iOS 操作系统分布（友盟指数）

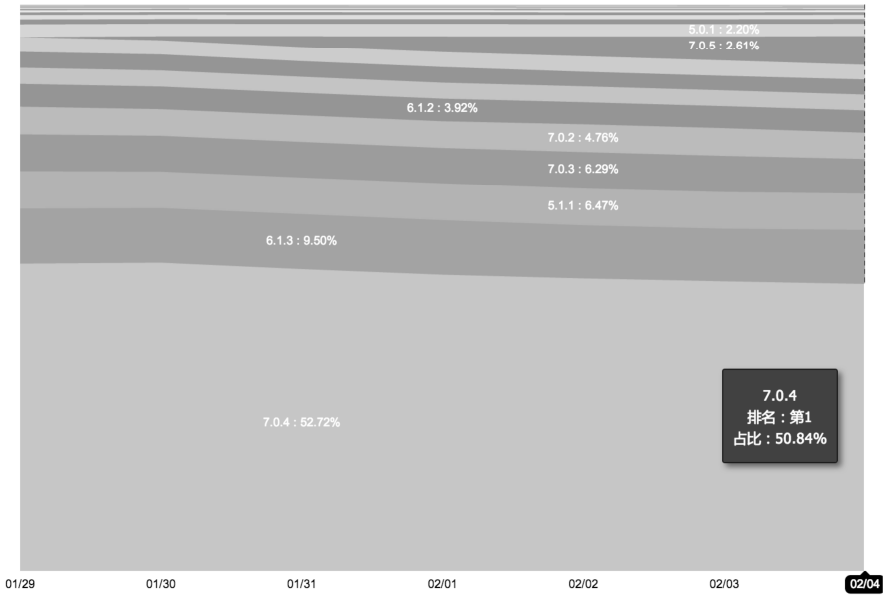



图 3-7　iOS 操作系统分布（WDM）

将以上 iOS 操作系统版本划分为：7.x、6.x、5.x、其他。在数据的选择上笔者统计了数值在 0.1%以上的数据，得到图 3-8 至图 3-10。

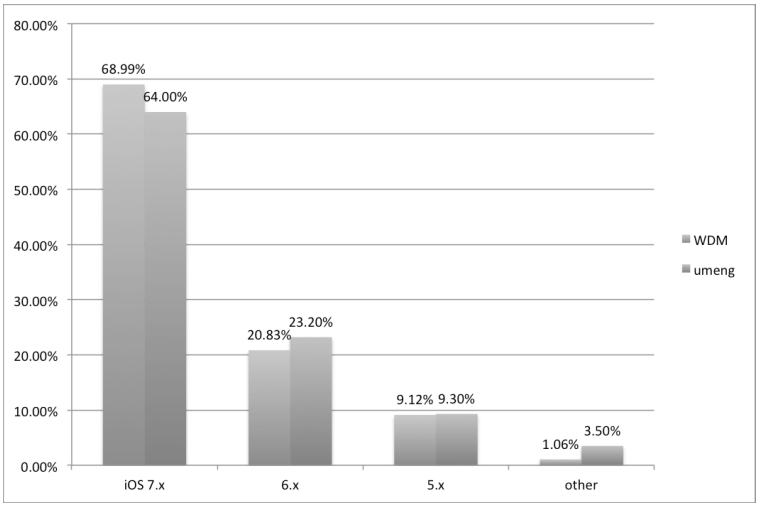



图 3-8　iOS 版本分布对比

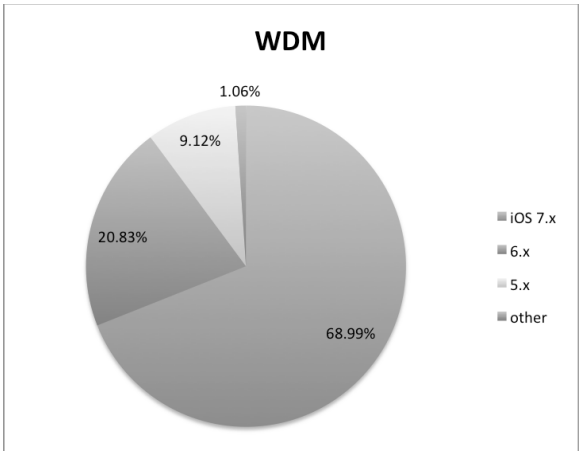



图 3-9 WDM 的 iOS 版本分布

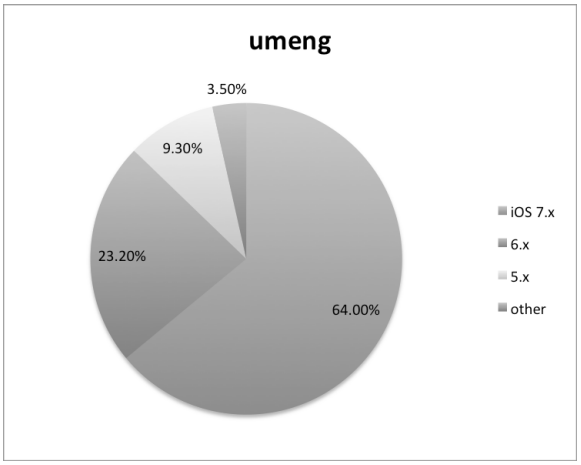



图 3-10 友盟指数的 iOS 版本分布

### 3.2.2.3 其他操作系统版本分布

国内的 Windows Phone 的占有量一直较少，来自 WDM 的“操作系统分布图”显示最近半年始终维持在 0.5%上下，如图 3-11 所示。其中的 aliyunos 是指“阿里云操作系统”。

国外的一份统计显示 Windows Phone 有着强劲的势头[10]：在英国 2013 年 2 月 Windows

[10] http://www.engadget.com/2013/04/01/windows-phone-sees-big-gain/

Phone 占有率已经达到 6.7%，而 2012 年只有 3%；意大利更是由 2012 年的 5.4%涨到了 2013 年的 13.1%；在国内，2013 年 2 月 Windows Phone 市场占有率也达到 1.4%。

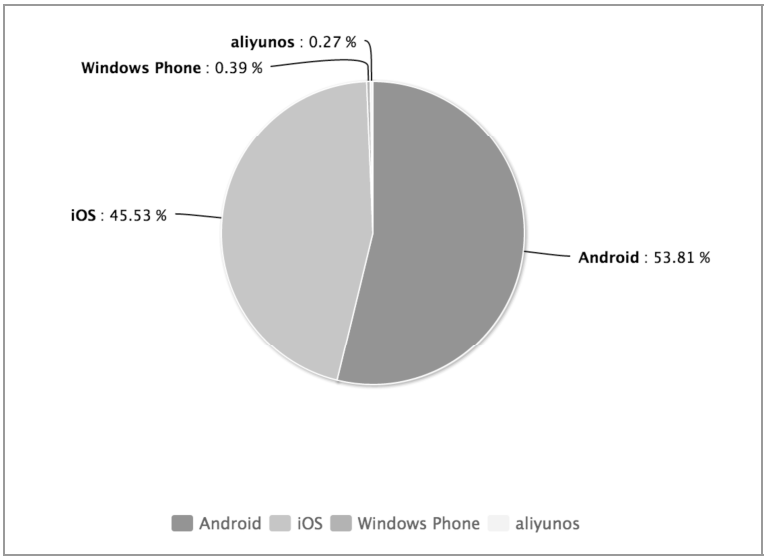



图 3-11　操作系统分布（WDM）

另外一份数据显示 2013 年 8 月欧洲 5 国（德、法、英、意、西）整体数据上 Windows Phone 占有率更是高达 9.2%[11]，如图 3-12 所示。

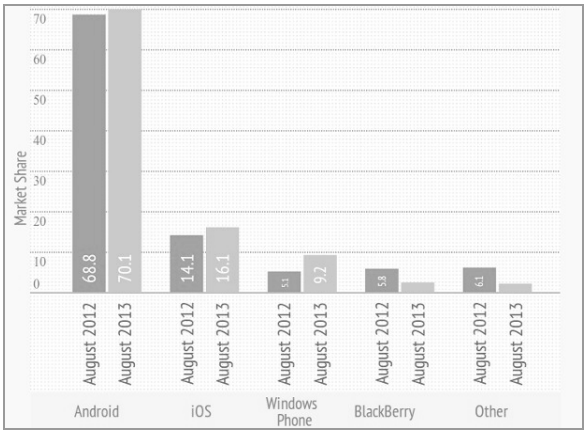



图 3-12　欧洲 5 国 2013 年 8 月移动 OS 市场占有率

[11] http://www.engadget.com/2013/09/30/windows-phone-market-share-europe/

最近公布的一份报告显示[12]，2013 年第 4 季度 Windows Phone 在欧洲 5 国的市场占有率继续增长：英、法均超过 11%，意大利更是达到了 17.1%，而在国内为 1.1%。

以上几份数据（见图 3-13、图 3-14 和图 3-15）同时也显示出 RIM（Black Berry）、Symbian 的全球市场占有率持续下跌，而 Bada 则从未起来过就已退出市场。此外，Tizen 的市场占有率数据在已查阅的公开数据源中未见数据。

| | 3 m/e Feb 2012<br>% | 3 m/e Feb 2013<br>% | Change<br>% |
|---|---|---|---|
| GB | 100,0% | 100,0% | 0,0 |
| iOS | 28,9 | 29,0 | 0,1 |
| Android | 48,3 | 58,3 | 10,0 |
| RIM | 16,8 | 5,1 | -11,7 |
| Symbian | 2,4 | 0,6 | -1,8 |
| Windows | 3,0 | 6,7 | 3,7 |
| Bada | 0,3 | 0,3 | 0,0 |
| Other | 0,3 | 0,1 | -0,2 |
| Germany | 100,0% | 100,0% | 0,0 |
| iOS | 21,6 | 18,7 | -2,9 |
| Android | 58,8 | 71,0 | 12,2 |
| RIM | 3,4 | 0,6 | -2,8 |
| Symbian | 9,0 | 2,0 | -7,0 |
| Windows | 6,3 | 6,8 | 0,5 |
| Bada | 0,7 | 0,4 | -0,3 |
| Other | 0,2 | 0,6 | 0,4 |
| Italy | 100,0% | 100,0% | 0,0 |
| iOS | 20,8 | 23,1 | 2,3 |
| Android | 51,0 | 57,1 | 6,1 |
| RIM | 3,4 | 2,0 | -1,4 |
| Symbian | 17,6 | 3,7 | -13,9 |
| Windows | 5,4 | 13,1 | 7,7 |
| Bada | 1,5 | 0,8 | -0,7 |
| Other | 0,3 | 0,2 | -0,1 |

图 3-13　2013 年 3 月欧洲 3 国 OS 系统市场占有率

[12] http://www.engadget.com/2014/01/27/kantar-report-android-up-samsung-down/

| | 3 m/e Feb 2012<br>% | 3 m/e Feb 2013<br>% | Change<br>% |
|---|---|---|---|
| US | 100,0% | 100,0% | 0,0 |
| iOS | 47,0 | 43,5 | -3,5 |
| Android | 45,4 | 51,2 | 5,8 |
| RIM | 3,6 | 0,7 | -2,9 |
| Symbian | 0,5 | 0,1 | -0,4 |
| Windows | 2,7 | 4,1 | 1,4 |
| Bada | 0,0 | 0,0 | 0,0 |
| Other | 0,8 | 0,4 | -0,4 |
| Australia | 100,0% | 100,0% | 0,0 |
| iOS | 31,9 | 32,5 | 0,6 |
| Android | 57,8 | 61,4 | 3,6 |
| RIM | 1,3 | 0,1 | -1,2 |
| Symbian | 4,8 | 1,2 | -3,6 |
| Windows | 1,7 | 3,4 | 1,7 |
| Bada | 0,1 | 0,1 | 0,0 |
| Other | 2,4 | 1,2 | -1,2 |
| Urban China | 100,0% | 100,0% | 0,0 |
| iOS | n/a | 25,8 | |
| Android | n/a | 68,7 | |
| RIM | n/a | 0,2 | |
| Symbian | n/a | 3,1 | |
| Windows | n/a | 1,4 | |
| Bada | n/a | 0,0 | |
| Other | n/a | 0,8 | |
| Mexico | 100,0% | 100,0% | 0,0 |
| iOS | 4,0 | 6,8 | 2,8 |
| Android | 25,9 | 55,8 | 29,9 |
| RIM | 33,2 | 20,2 | -13,0 |
| Symbian | 29,2 | 9,5 | -19,7 |
| Windows | 5,2 | 5,9 | 0,7 |
| Bada | 0,7 | 1,5 | 0,8 |
| Other | 1,7 | 0,2 | -1,5 |

图 3-14　2013 年 3 月多国移动 OS 的市场占有率

| Germany | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 69.0 | 75.4 | 6.4 |
| BlackBerry | 1.1 | 0.5 | -0.6 |
| iOS | 21.7 | 17.3 | -4.4 |
| Windows | 3.4 | 5.9 | 2.5 |
| Other | 4.8 | 0.9 | -3.9 |

| USA | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 46.2 | 50.6 | 4.4 |
| BlackBerry | 0.9 | 0.4 | -0.5 |
| iOS | 49.7 | 43.9 | -5.8 |
| Windows | 2.4 | 4.3 | 1.9 |
| Other | 0.8 | 0.8 | 0.0 |

| GB | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 54.4 | 54.9 | 0.5 |
| BlackBerry | 6.4 | 3.2 | -3.2 |
| iOS | 32.4 | 29.9 | -2.5 |
| Windows | 5.9 | 11.3 | 5.4 |
| Other | 0.9 | 0.6 | -0.3 |

| China | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 73.7 | 78.6 | 4.9 |
| BlackBerry | 0.0 | 0.1 | 0.1 |
| iOS | 21.2 | 19.0 | -2.2 |
| Windows | 0.9 | 1.1 | 0.2 |
| Other | 4.2 | 1.3 | -2.9 |

| France | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 61.0 | 65.9 | 4.9 |
| BlackBerry | 5.1 | 1.6 | -3.5 |
| iOS | 23.7 | 20.3 | -3.4 |
| Windows | 5.0 | 11.4 | 6.4 |
| Other | 5.1 | 0.8 | -4.3 |

| Australia | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 56.0 | 57.2 | 1.2 |
| BlackBerry | 1.0 | 0.8 | -0.2 |
| iOS | 38.5 | 35.2 | -3.3 |
| Windows | 3.0 | 5.2 | 2.2 |
| Other | 1.5 | 1.7 | 0.2 |

| Italy | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 54.2 | 66.2 | 12.0 |
| BlackBerry | 2.6 | 1.8 | -0.8 |
| iOS | 23.1 | 12.8 | -10.3 |
| Windows | 12.7 | 17.1 | 4.4 |
| Other | 7.4 | 2.1 | -5.3 |

| LatAm 3 (BR, BX, AR) | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 61.6 | 83.5 | 21.9 |
| BlackBerry | 10.3 | 2.8 | -7.5 |
| iOS | 4.4 | 4.3 | -0.1 |
| Windows | 6.8 | 4.9 | -1.8 |
| Other | 17.0 | 4.5 | -12.5 |

| Spain | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 85.9 | 86.2 | 0.3 |
| BlackBerry | 2.4 | 0.2 | -2.2 |
| iOS | 7.3 | 6.7 | -0.6 |
| Windows | 1.2 | 5.6 | 4.4 |
| Other | 3.2 | 1.3 | -1.9 |

| EU5 | 3 m/e Dec 2012 | 3 m/e Dec 2013 | % pt. Change |
|---|---|---|---|
| Android | 62.9 | 68.6 | 5.7 |
| BlackBerry | 3.7 | 1.5 | -2.2 |
| iOS | 23.7 | 18.5 | -5.2 |
| Windows | 5.6 | 10.3 | 4.6 |
| Other | 4.0 | 1.1 | -3.0 |

图 3-15 2013 年 Q4 多国移动 OS 市场占有率

### 3.2.2.4 操作系统趋势

操作系统的版本趋势数据可以在制定最后分级时有更好的前瞻性。笔者引用了以下几份不同的趋势数据，如图 3-16、图 3-17 和图 3-18 所示。

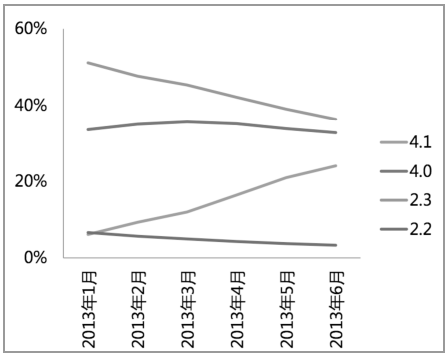



图 3-16　Android 版本变化趋势[13]

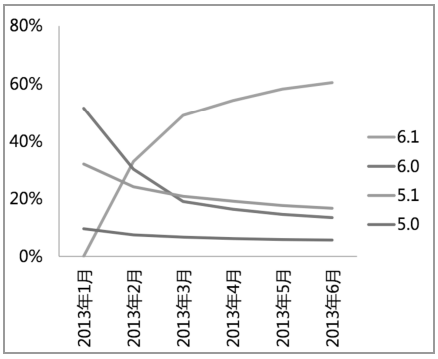



图 3-17　iOS 版本变化趋势

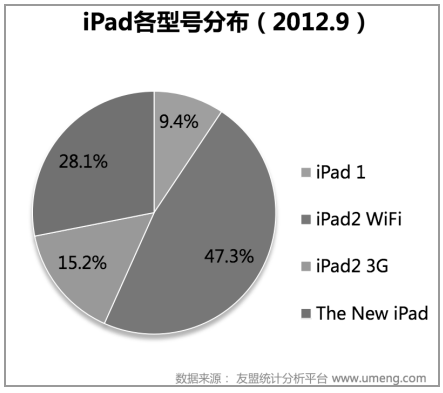



图 3-18　iPad 各型号分布[14]

[13] 《友盟 2013 年中国移动互联网上半年报告 2013.09》http://www.umeng.com/umengdata_reports。

[14] 《友盟年度报告:移动互联网重塑用户生活 2013.03》 http://www.umeng.com/umengdata_reports。

#### 3.2.2.5 操作系统分级

加底纹部分是笔者在 2014 年 2 月更新这份数据（见表 3-1）时，相对 2013 年 5 月（见表 3-2）的重要变化。

- iOS 7.x 从无到有并成为 A 级，iOS 6.x 从 A 级退至 B 级。
- Android 4.4.x、Windows Phone 进入 C 级观察。
- Android 2.2.x 和 iOS 4.x 退出历史舞台。

表 3-1 操作系统分级（2014.2）

| 级别 | 操作系统 | | |
|---|---|---|---|
| A 级 | iOS 7.x | Android 4.3.x | Android 2.3.x |
| B 级 | iOS 6.x | Android 4.1.x | Android 4.0.x |
| C 级 | iOS 5.x | Android 4.4.x | Windows Phone |

表 3-2 操作系统分级（2013.5）

| 级别 | 操作系统 | |
|---|---|---|
| A 级 | Android 4.x | Android 2.3.x |
| | iOS 6.x | |
| B 级 | iOS 5.x | |
| C 级 | Android 2.2.x | iOS 4.x |

### 3.2.3 屏幕分辨率分级

#### 3.2.3.1 Android 屏幕分辨率

Android 的屏幕分辨率众多（截图日期 20140205），如图 3-19 和图 3-20 所示。

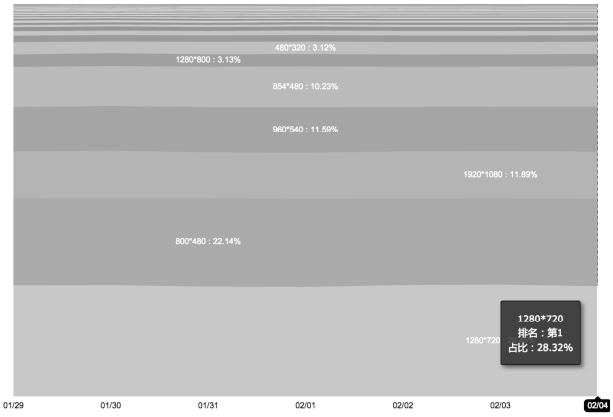



图 3-19 Android 屏幕分辨率（WDM）

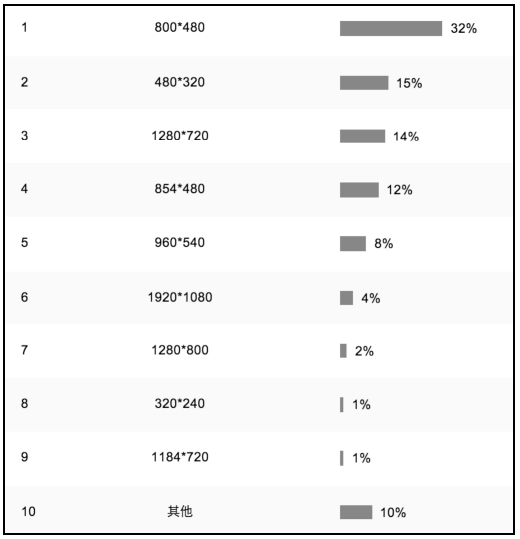



图 3-20　Android 屏幕分辨率（友盟指数）

相对 iOS 而言，Android 设备的屏幕分辨率的种类极其多。尽管如此，工程中合理设置页面的 meta Viewport 属性[15]后并且对页面进行合理布局（主体内容固定宽度 320px）能够大大简化页面需要适配的分辨率种类。笔者将 Android 的屏幕分辨率划分为：320+（480/320/540 等）、640+（800/768/720/640 等）和其他，如图 3-21 和图 3-22 所示。其中“640+”这一组设置 Viewport 后 CSS 计算时使用的宽度是 320px+，需要双倍宽度的图片。

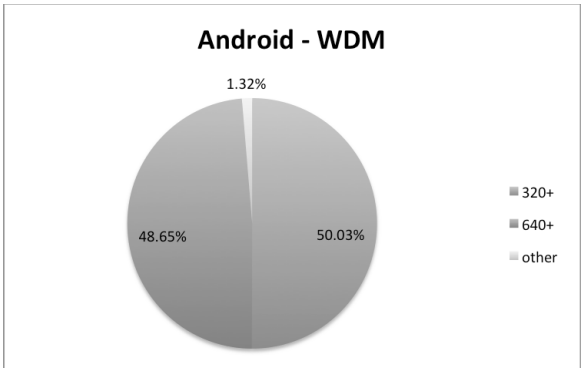



图 3-21　Android 屏幕分辨率（WDM）

[15] 参见本书第 2 章的“Viewport”一节。

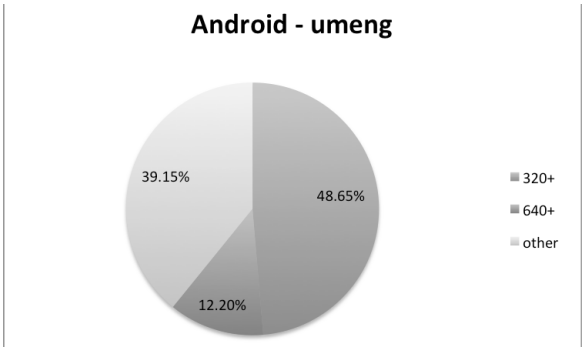



图 3-22 Android 屏幕分辨率（友盟指数）

### 3.2.3.2 iOS 屏幕分辨率

iOS 屏幕分辨率相对统一（截图日期 20140205），如图 3-23 和图 3-24 所示。

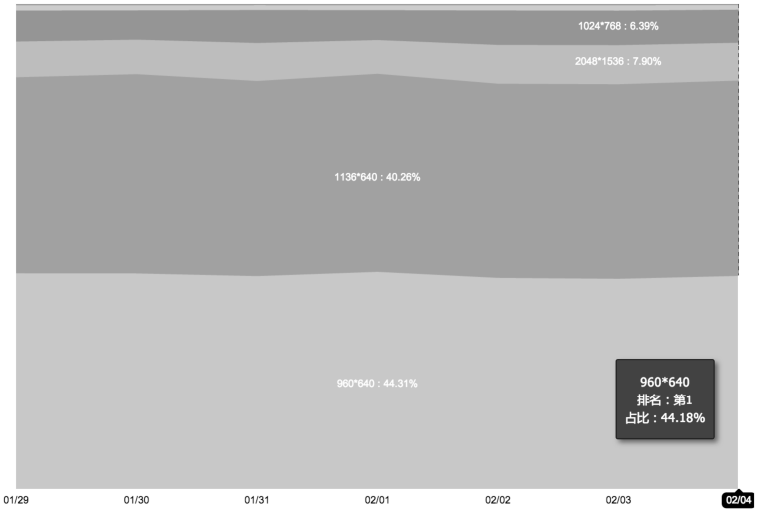



图 3-23 iOS 屏幕分辨率（WDM）

| | | |
|---|---|---|
| 1 | 960*640 | 45.1% |
| 2 | 1136*640 | 34.0% |
| 3 | 1024*768 | 10.1% |
| 4 | 2048*1536 | 9.8% |
| 5 | 480*320 | 1.0% |

图 3-24 iOS 屏幕分辨率（友盟指数）

从以上数据整理出如下 iPhone 和 iPad 的屏幕分辨率分布图表，如图 3-25 至图 3-28 所示。

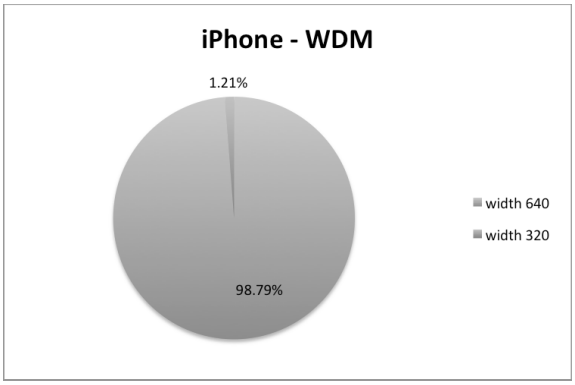



图 3-25　iPhone 屏幕分辨率（WDM）

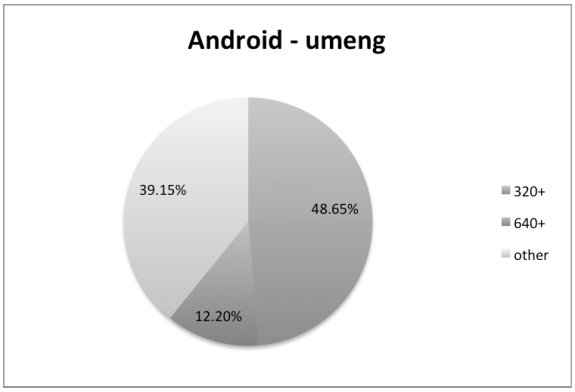



图 3-26　iPhone 屏幕分辨率（友盟指数）

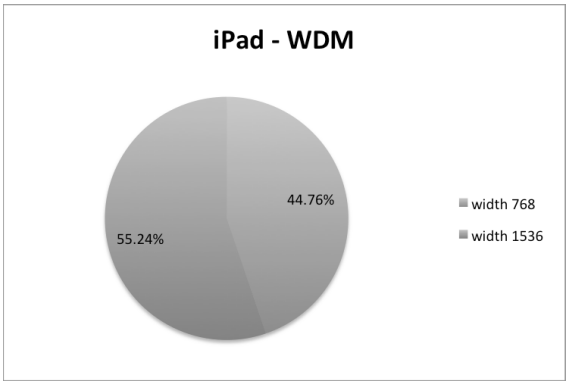



图 3-27　iPad 屏幕分辨率（WDM）

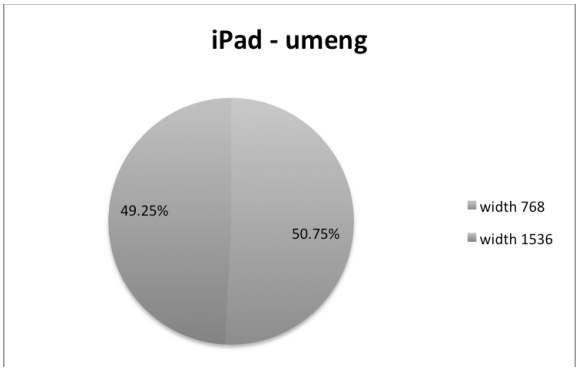



图 3-28 iPad 屏幕分辨率（友盟指数）

iPhone 4+（屏宽 640）占有率在 2012 年第 3 季就超过了 95%，之后 iPhone 5 的占有率又在持续提升，所以目前 iPhone 的屏幕分辨率就定为 640；从 iPad 3（the new iPad）开始，iPad 启用 retina 屏，分辨率达到了 2048×1536，iPad 3 以及之后的 iPad 4 已经有了相当的占有率，是否考虑支持 retina 还有待讨论。

再来看下 iOS 设备分布情况（截图日期 20140205），如图 3-29 所示。

| | | |
|---|---|---|
| 1 | iPhone 4S | 23.1% |
| 2 | iPhone 5 | 20.4% |
| 3 | iPhone 4 | 19.3% |
| 4 | iPhone 5s | 12.0% |
| 5 | iPad 2 | 5.4% |
| 6 | iPad mini | 5.0% |
| 7 | iPad 4 | 4.7% |
| 8 | The New iPad | 2.7% |
| 9 | iPad Air | 2.5% |
| 10 | iPhone 5c | 2.0% |
| 11 | iPad mini 2 | 1.1% |

图 3-29 iOS 设备分布（友盟指数）

iOS 设备分布数据来自友盟 2012 年年度数据报告，如图 3-30 所示。

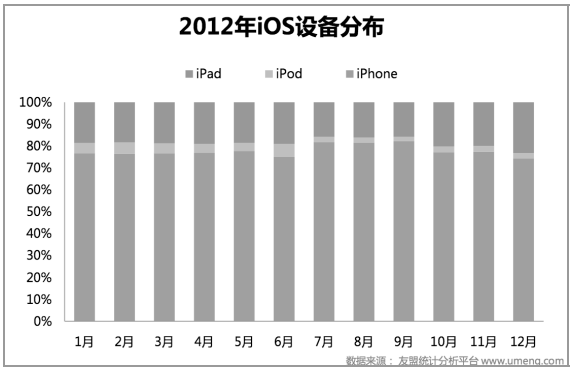



图 3-30　iOS 设备分布趋势

iOS 设备版本分布数据来自友盟 2012 年第 3 季度数据报告 2012Q3（注：友盟 Q4 报告没有这份数据）如图 3-31 和图 3-32 所示。

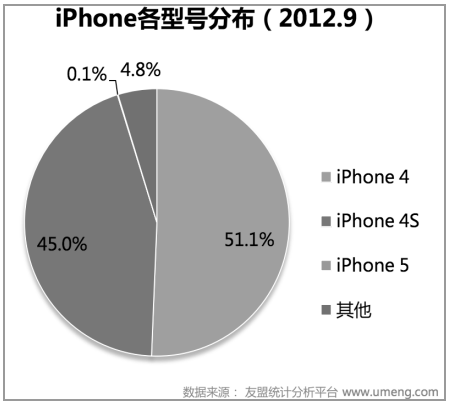



图 3-31　iPhone 型号分布

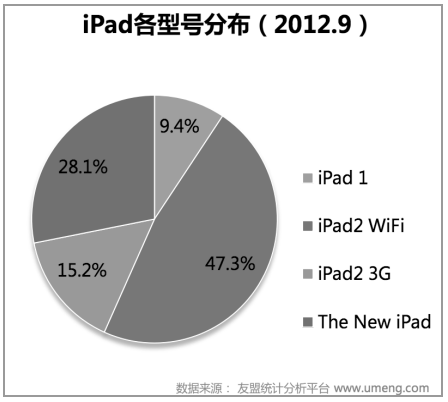



图 3-32　iPad 型号分布

#### 3.2.3.3 屏幕分辨率分级

最终得到表 3-3，其中“Android Pad”是相对 2013 年 5 月的新增内容。

表 3-3 屏幕分辨率分级

| 级别 | 环境 | 典型分辨率（典型设备） |
|---|---|---|
| A 级 | iPhone Viewport 320 | Viewport 标准化后的页面宽度为 320px<br>640*960（iphone 4/4s）、640*1136（iPhone 5+） |
| | iPad Viewport 768 | Viewport 标准化后的页面宽度为 768px<br>1536*2048（iPad 3+ Retina）、768*1024（iPad 2/mini） |
| | Android Viewport 320+ | Viewport 标准化后的页面宽度为 320px、360px、374px、400px 等<br>普通：480*800、480*854、320*480、540*960（HTC g18/g19）<br>Retina：640*960（魅族 MX）、640*1136、720*1280（Note 2、Galaxy S3）、1080*1920（Note 3、Nexus 5）、768*1280（Nexus 4）、800*1280（Note 1） |
| B 级 | | |
| C 级 | Android Pad | 800*1280（Galaxy Note 10.1）、1600*2560（Galaxy Note 2014）、768*1024 |

### 3.2.4 浏览器分级

#### 3.2.4.1 Android 浏览器

无论从首选率还是品牌使用率，UC 浏览器、QQ 浏览器、默认浏览器属于第一阵营，远远甩开其他浏览器，如图 3-33 和图 3-34 所示。

《2012 年中国手机浏览器用户行为研究报告》（CNNIC）[16]，参考第 4 章。

[16] 《2012 年中国手机浏览器用户行为研究报告》
http://cnnic.net.cn/hlwfzyj/hlwxzbg/ydhlwbg/201211/P020121116518463145828.pdf。

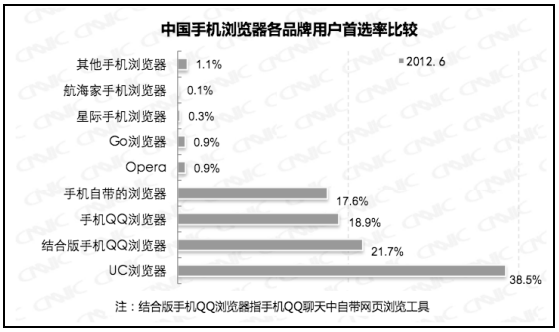



图 3-33　中国手机浏览器品牌用户首选率

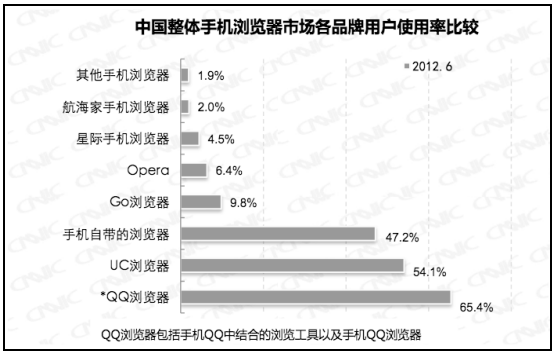



图 3-34　中国手机浏览器品牌用户使用率

### 3.2.4.2　iOS 浏览器

已查阅的国内的各种数据报告中均未提及，因此参考了 Net Application 201303 Mobile Browser Market Share，综合 iOS 和 Android 操作系统的市场占有率，可以近似估计 iOS 设备上 Safari 占有率，如图 3-35 所示。

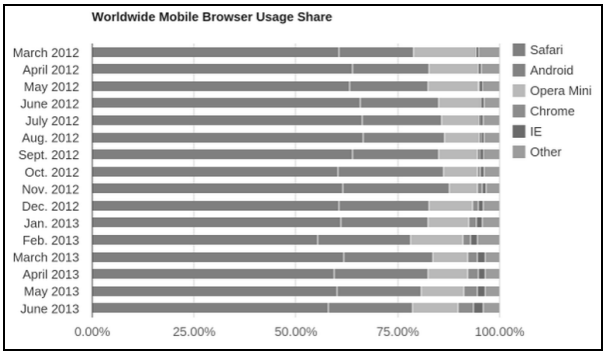



图 3-35　全球移动浏览器使用率（Net Application）

#### 3.2.4.3 浏览器分级

综合以上 Android、iOS 浏览器数据得到表 3-4。

表 3-4 浏览器分级

| 级别 | 操作系统 | |
|---|---|---|
| A 级 | Android UC 浏览器 † | Android 系统自带浏览器 |
| | Android QQ 浏览器 † | iOS Safari † |
| B 级 | Android/iOS Chrome † | |
| C 级 | Android/iOS Opera † | |
| | Android Go † | |

### 3.2.5 MGBS

综合前面几节的讨论，我们得到一张完整的 MGBS，包括了：浏览器、操作系统、屏幕分辨率，如表 3-5 所示。

表 3-5 MGBS

| 级别 | 浏览器 | |
|---|---|---|
| A 级 | Android UC 浏览器 † | Android 系统自带浏览器 |
| | Android QQ 浏览器 † | iOS Safari † |
| B 级 | Android/iOS Chrome † | |
| C 级 | Android/iOS Opera † | |
| | Android Go † | |

| 级别 | 操作系统 | |
|---|---|---|
| A 级 | Android 4.x | Android 2.3.x |
| | iOS 6.x | |
| B 级 | iOS 5.x | |
| C 级 | Android 2.2x | iOS 4.x |

| 级别 | 测试宽度 | 典型分辨率（典型设备） |
|---|---|---|
| A 级 | Android 480 | 480*800、480*854 |
| | iPhone 640 | 640*960(iPhone 4/4s)、640*1136(iPhone 5) |
| | iPad 768 | 768*1024(iPad 2/mini) |
| B 级 | Android 320 | 320*480 |
| | Android 720 | 720*1280(Galaxy s3) |
| | iPad 1536 | 1536*2048(iPad 3/4) |
| | Android 540 | 540*960(HTC g18/g19) |
| C 级 | Android 800 | 800*1280(Galaxy Tab、Note 10.1) |

**描述**

1. x 当前最新子版本。
2. † 当前正式最新版（非 BETA）。
3. 测试基准分为 3 级，A 级优先级最高，B 级次之，C 级为可选。
4. B 级也是可选，根据不同业务的实际情况选择。
5. 每季度更新一次（初期更新频度会加快）。
6. 暂定不支持设备横屏（landscape）。
7. Android QQ 浏览器使用系统 WebView，表现和系统自带浏览器接近（可能还是会有差异）。

**分级说明**

1. A 级为优先级最高，A 级要求测试通过所有的测试用例。前端需要在提交测试之前，按照上线标准进行充分的自测。
2. B 级是对 A 级的补充。B 级和 A 级都需要通过所有的测试用例，区别就是，前端不需要在 B 级上进行自测，而是交给测试完成。
3. C 级优先级比较低，各业务线测试可以进行抽样测试，不需要做到测试用例完全覆盖，也不需要保持样式完全一致。如果遇到具体的 bug，需要和测试一起评定 bug 的严重性。在双方认可的情况下，允许不解决 C 级以上的 bug 而发布代码。

**渐进增强和平稳退化**

**渐进增强：**在高级浏览器上，作为前端，需要考虑如何利用浏览器提供的资源，提供更好的用户体验。

**平稳退化：**平稳退化不是不做某些功能，而是提供一些更适合的功能，更适合的交互给适合的平台。不让繁杂的功能由于在不适合的平台上使用，而成为一个负担。

#### 3.2.5.1　基准落地

为了便于前端和测试同学理解，并且去除不重要的组合，最终落地的基准如表 3-6 所示。

表 3-6 MGBS 落地

| 级别 | 浏览器 |
| --- | --- |
| A 级 | Android 4.x 系统自带浏览器† |
| | Android 2.3.x 系统自带浏览器† |
| | Android 4.x UC 浏览器† |
| | Android 4.x QQ 浏览器† |
| | iOS 6.x Safari † |
| B 级 | Android 4.x Chrome† |
| | iOS 6.x Chrome † |

| 设备 | 测试宽度 | 参考分辨率 |
| --- | --- | --- |
| Android | 480 | 480*800、480*854 |
| iPhone | 640 | 640*960(iPhone 4/4s)、640*1136(iPhone 5) |
| iPad | 768 | 768*1024(iPad 2/mini) |

#### 3.2.5.2 遗留问题

限于时间，仍然有一些细节问题在本书完结之时未去深究。

1. iPad retina（屏宽 1536）支持。
2. Android 屏宽 720/800 支持。
3. 所有移动设备的横屏支持。
4. Android 2.3.x 与 4.x 的默认浏览器差异，以及可接受的退化处理范围。
5. Android QQ 浏览器是否仍然在使用的 WebView，是否有必要放在 A 级。
6. Android QQ 浏览器有独立和 QQ 聊天内置两种形式，是否有差异，待调研。
7. Windows Phone 支持，本书写作过程中，国内 Windows Phone 的占有率始终徘徊在 0.5%上下。

## 3.3 GTE

更进一步，可以建立一个包含 GBS 和 MGBS 且面向未来的分级目标环境（GTE）[17]。

1. 过去：GBS（Graded Browser Support）、MGBS（Mobile GBS）是分别针对 PC Browser 和 Mobile Browser 环境制定的分级浏览器标准。

[17] YUI Target Environments：http://yuilibrary.com/yui/environments/

2. 现状：页面已经运行在 PC 浏览器、Mobile 浏览器、Mobile 客户端，并且这个趋势现在更加明显，客户端中的 WebView 已经成为业务中的重要的移动浏览器。
3. 未来：之后还可能出现页面运行在 TV 端，页面模板的渲染代码可能在服务器端执行（Node.js）。

### 3.3.1　分层设计

GTE 的整体结构分为 3 层。

1. 用户层（xGTE）。GTE 的用户可能需要适合自己情况的 GTE。
2. 核心层（GTE）。GTE 的完整版本，各个 xGTE 依赖 GTE。
3. 数据层（GTED）。GTE 的数据来源。

### 3.3.2　核心层

表 3-7 可以看作是 GBS 和 MGBS 的合并版本，另外添加了机顶盒（TV 端）、Node.js 环境。

几点说明如下。

1. 单元格加底纹代表新增（相对于更早之前的 MGBS 和 GBS）。
2. 新增对机顶盒（TV 端）的 C 级监控。
3. Windows Phone、Node.js 进入 C 级统计。
4. iOS 7.x 更新速度非常快，目前已超过 60%，iOS 6.x 持续下降。
5. Android 2.2.x、iOS 4.x 由于占有率连续两次统计低于 1%，已经退出 C 级。

表 3-7　GTE

| 级别 | 浏览器 | | | |
|---|---|---|---|---|
| A 级 | iOS Safari † | Android 内置浏览器 † | Android UC 浏览器 † | Android QQ 浏览器 † |
| | iOS 客户端 | | Android 客户端 | |
| | Windows IE 8 | Windows IE 6 | Windows Chrome † | Windows IE 10 |
| B 级 | iOS Chrome † | Android Chrome † | | |
| | Windows IE 7 | Windows IE 9 | iPad Safari | Windows 360 安全浏览器 |

续表

| 级别 | 浏览器 | | | |
|---|---|---|---|---|
| C 级 | Android Opera † | iOS Opera † | Windows Phone IE † | Android Pad 内置浏览器 † |
| | Windows 遨游浏览器 † | Windows 猎豹浏览器 † | Windows QQ 浏览器 † | Windows Safari † |
| | Windows Firefox † | Windows 淘宝浏览器 † | Windows IE 11 | |
| | 机顶盒内置浏览器 † | Node.js † | | |
| 级别 | 操作系统 | | | |
| A 级 | iOS 7.x | Android 4.3.x | Android 2.3.x | Windows |
| B 级 | iOS 6.x | Android 4.1.x | Android 4.0.x | |
| C 级 | iOS 5.x | Android 4.4.x | Windows Phone | OS X |

| 级别 | 环境 | 典型分辨率（典型设备） |
|---|---|---|
| A 级 | iPhone Viewport 320 | Viewport 标准化后的页面宽度为 320px<br>640*960（iPhone 4/4s） 640*1136（iPhone 5+） |
| | iPad Viewport 768 | Viewport 标准化后的页面宽度为 768px<br>1536*2048（iPad 3+ retina）、768*1024（iPad 2/mini） |
| | Android Viewport 320+ | Viewport 标准化后的页面宽度为 320px、360px、374px、400px 等<br>普通：480*800、480*854、320*480、540*960 （HTC g18/g19）<br>Retina：640*960（魅族 MX）、640*1136、720*1280（Note 2、Galaxy S3）、1080*1920（Note 3、Nexus 5）、768*1280（Nexus 4）、800*1280（Note 1） |
| | Windows 1280+ | 含 1280px、1360px、1440px、1690px 及以上 |
| B 级 | Windows 1024 | |
| C 级 | Android Pad | 800*1280（Galaxy Note 10.1）、1600*2560（Galaxy Note 2014）、768*1024 |

### 3.3.3　数据层

数据层最终产出是一张如图 3-36 和图 3-37 所示的数据报表。读者可以下载到这张报表[18]，根据需要更改或扩展报表结构并用真实数据填充。

报表被设计为：

- 三级数据
  - 一级，指标（操作系统、屏幕分辨率、浏览器）
  - 二级，环境（Mobile、PC、TV）
  - 三级，版本
- 对应三级数据的 PV、UV
- 穿插了 iPad 或一些小版本的合并信息
- 大的色块区分指标，颜色深浅区分级别

| 一级 | 二级 | 三级 | 版本 | PV | 三级占比 | 二级占比 | 一级占比 | UV | 三级占比 | 二级占比 | 一级占比 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 操作系统 | Mobile | iOS | 7.x | | 68.99% | | | | | | |
| | | | 6.x | | 20.83% | | | | | | |
| | | | 5.x | | 9.12% | | | | | | |
| | | | 4.x | | 0.53% | | | | | | |
| | | | 和 | 0.00% | 99.47% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| | | | iPad | | | | | | | | |
| | | Android | 4.4.x | | 0.37% | | | | | | |
| | | | 4.2.x-4.3.x | | 34.04% | | | | | | |
| | | | 4.1.x | | 35.49% | | | | | | |
| | | | 4.0.x | | 17.84% | | | | | | |
| | | | 2.3.x | | 10.75% | | | | | | |
| | | | 2.2.x | | 0.42% | | | | | | |
| | | | 和 | 0.00% | 98.91% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| | | | 4.x 和 | 0.00% | 87.74% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| | | | Pad | | | | | | | | |
| | | Windows Ph | 和 | | | 0.00% | | | | | |
| | | aliyunos | 和 | | | 0.00% | | | | | |
| | | 和 | | 0 | | 0 | 0 | 0 | 0 | 0 | 0 |
| | PC | Windows | 8 | | | | | | | | |
| | | | 7 | | | | | | | | |
| | | | xp | | | | | | | | |
| | | | 和 | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| | | OSX | 和 | | | 0.00% | | | | | |
| | | 和 | | 0 | | 0 | 0 | 0 | 0 | 0 | 0 |
| 分辨率 | Mobile | iOS | 640 | | 84.27% | | | | | | |
| | | | 320 | | 1.03% | | | | | | |
| | | | 1536 | | 8.12% | | | | | | |
| | | | 768 | | 6.58% | | | | | | |
| | | | 和 | 0.00% | 100.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| | | Android | 600+ | | 48.65% | | | | | | |

图 3-36　GTE 报表模板

[18] GTE 报表模板：http://luics.github.io/cew/GTE.xlsx。

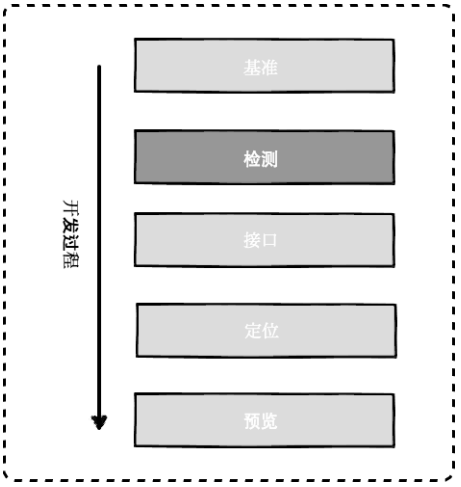
| 子公司 | 总PV | 总UV |
| --- | --- | --- |
| 公司A | 0 | 0 |
| 公司B | 0 | 0 |
| 公司C | 0 | 0 |
| 公司D | 0 | 0 |
| … | | |

| 其他环境 | PV | UV |
| --- | --- | --- |
| Node.js 环境 | | |
| … | | |

图 3-37 GTE 数据汇总和其他环境

# 4

# 检测




由 1.2 节例子中需要根据访问来源（Mobile 或 PC）来决定返回的页面，这就遇到了终端检测这个问题。

## 4.1 终端

### 4.1.1 什么是终端

本书开篇提到过终端和设备的差异，那么终端更精确的定义是什么呢？

从常规意义来说：

> 终端，是一台电子计算机或者计算机系统，用来让用户输入数据，及显示

其计算结果的机器。

对 Web 应用开发者来说：

拥有独立运行时的 Web 解析容器都是一个终端，既可以是一台独立硬件设备，也可能是设备上的某个运行环境，如 iPhone 5s 的 Chrome 浏览器。

### 4.1.2　分类

#### 4.1.2.1　设备类型

手机（Phone）、平板（Tablet/Pad）、个人电脑（PC）是目前最流行的 3 类终端设备。处于变革中的 TV 也有成为第 4 类终端设备的潜力。

根据在人群中的流行程度，目前的智能设备可以分为以下几类。

- 手机：Phone
- 平板：Tablet/Pad
- 个人电脑：PC

#### 4.1.2.2　操作系统

参考第 3 章“基准”和已公开的数据，来自于三家厂商的操作系统占据着终端的主流市场。

- Apple：OS X（桌面系统）、iOS（Phone 和 Pad）。
- Google：份额后来居上的 Android（Phone 和 Pad）。
- Microsoft：覆盖手机、平板、个人电脑的 Windows 系列。

## 4.2　终端检测

### 4.2.1　场景

1.2 节曾提到下面这个例子：手机端展现 PC 端网页的体验是多么糟糕，如图 4-1 所示！


图 4-1 不合适的浏览体验

解决这个问题的一种做法是由开发者在后端或前端识别终端特性，然后跳转到合适的版本。这种“页面适配设备”只是终端检测众多理由中的一个，在实际的项目开发中，需要终端检测的场景还有很多。

- 显示或隐藏特定内容。
- 加载特定的静态资源样式、脚本。
- 静态扫描修改文件源。
- 返回指定图片质量/大小的图片。

## 4.2.2 原理

用户代理（User Agent，简称 UA）是最常用的终端检测方式。

> 用户代理（User Agent）指的是代表使用者行为的软件（软件代理程序）所提供的对自己的一个标识符。

图 4-2 是一个 HTTP 请求头信息的一部分。

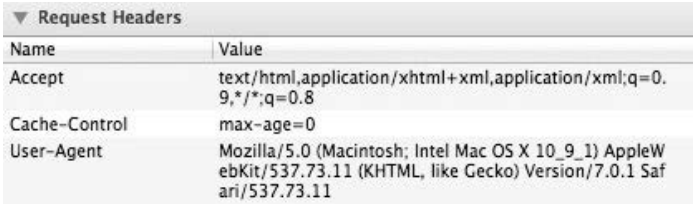



图 4-2　HTTP 请求头信息的一部分

User-Agent 中包括：

- “Macintosh; Intel Mac OS X 10_9_1”指定操作系统。
- “AppleWebKit/537.73.11 （KHTML, like Gecko） ”指定浏览器内核。
- “Version/7.0.1 Safari/537.73.11”指定浏览器软件版本和内核版本。

对这个 User-Agent 分析可以得到一些简单的信息：这个请求来自于 Mac 系统上内核为 WebKit 的 Safari 浏览器。

遗憾的是，尽管 HTTP 1.1（RFC2616）约定了 User-Agent[1]的格式，但现实世界中，基本没有浏览器厂家遵循。尽管如此，好在各浏览器都有一套相对稳定的 User-Agent 私有格式，仍然有可能去检测它。

操作系统、浏览器内核、软件版本是终端检测中最重要的信息，应用场景也较多，如：

- 把 Android 和 iOS 用户引导到对应的应用商店。
- 获知网页是否运行在移动 App 的 WebView 内。
- 提示 IE 6 用户升级浏览器。

随着应用程序功能的增多，检测硬件信息的需求也愈发频繁，如：

- 显示屏幕的宽、高。
- 物理信息，如屏幕的物理尺寸。
- 传感器支持，如加速计、陀螺仪、摄像头、传声器等。这些特性需要在终端检测的基础上获取对应的设备信息才能实现。

[1] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.43

### 4.2.3 实现

终端检测的方法很多，不同的应用场景下也有不同的实现。图 4-3 是一种典型的基于 User-Agent 的服务器端实现——MED 架构。

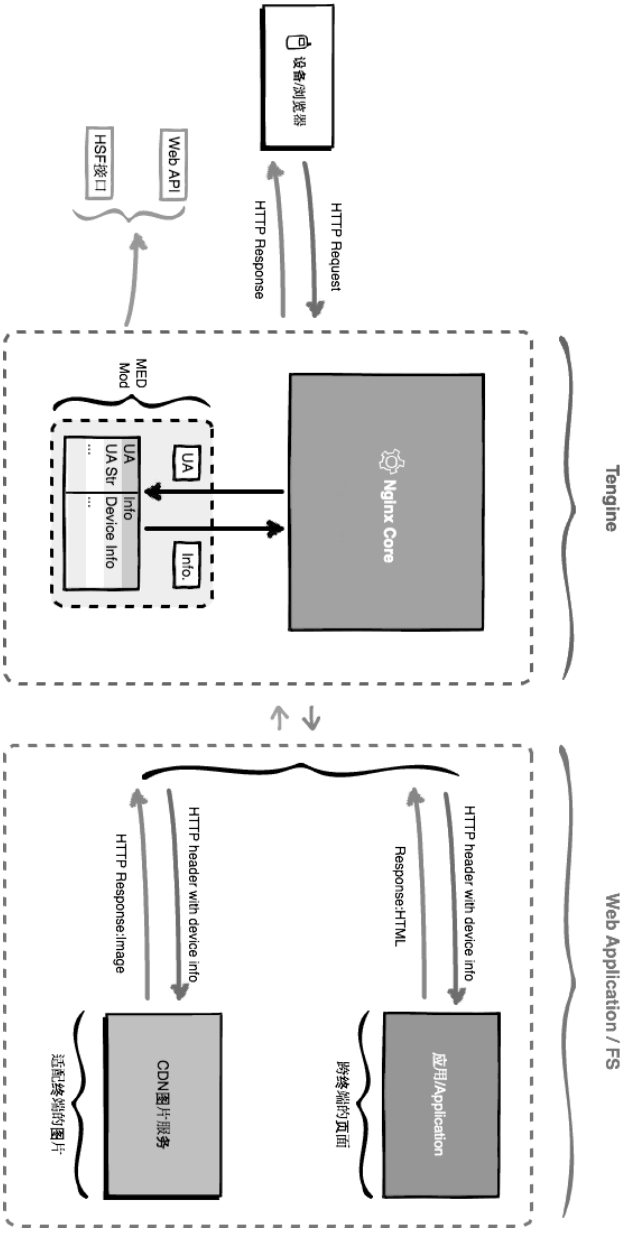



图 4-3 MED 架构

- 终端检测使用 User-Agent。
- 部署于应用服务器之前，将检测到的信息置于 HTTP 头部，可让所有应用服务器访问到。
- 引入“UA 库”，可以通过 UA 检索出设备信息，在终端检测之外带来了更多设备信息。
- 终端检测对前端、后端的开发人员透明。
- 终端检测除了应用于页面级跳转，还应用于优化图片等静态资源的响应。

总的来说，这种实现方式更适合大规模的应用场景，结构也更为复杂，对稳定性要求也更高，带来的好处是检测这件事对前后端开发人员是透明的，服务器端控制也有更好的灵活性。

## 4.3　遗留问题

### 4.3.1　硬件信息

iPhone 4 和 iPhone 5s 如果都安装了 iOS 7.0，那么 Mobile Safari 的 UA 是完全相同的，这样想从 UA 再获取硬件信息就不精确了。

> Mozilla/5.0 （iPhone; CPU iPhone OS 7_0_2 like Mac OS X）
> AppleWebKit/537.51.1（KHTML, like Gecko）Version/7.0 Mobile/11A4449d
> Safari/9537.53

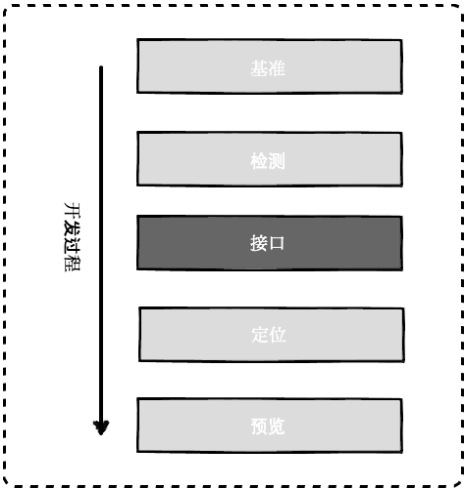
Android 设备间差异更大，这个问题在 Android 上会更明显。

### 4.3.2　更精准的终端检测

前面也说到各浏览器都有一套相对稳定的 User-Agent 私有格式，但毕竟是私有格式，并没有严格规范。即使对同一个浏览器也很难总结出一套能覆盖所有 case 的规则集合，这会影响检测的精度。

引入机器学习是一种已被证实有效的解决办法，基本思路是将 UA 作为特征（或进一步提取特征），采集一定数量的样本学习后用于终端检测。SVM 的改进算法和神经网络都有实际应用。

# 5 接口




1.2 节例子中前后端通过一套数据接口通信，这样能减少后端的重复开发，除此之外还有诸多益处。本章会从接口实现流程复用的角度来讨论这个问题，并介绍了一个有趣的项目“IF”来解决接口中的一些问题。2014 年 2 月，“IF——端到端的接口规范”入选“2013 阿里技术那些事”前端交互与设计领域的十大事件。

## 5.1 跨终端流程复用

跨终端时代的到来，让我们看到前后端解耦一个潜在的优势：**流程复用**。

### 5.1.1 示例 1

以虚拟电商公司 VCom 为例，VCom 有一套购物主流程：登录、商品列表、商品详

情、购物车、结算。假如前后端不进行解耦，那么在跨终端的今天，VCom 需要为 PC、Phone Web、Pad Web、Phone App、Pad App 开发多套产品。如果 PC Web 和 Pad Web 使用一套，Phone App 至少覆盖 iOS 和 Android，Pad App 至少覆盖 iPad，也就是至少开发 5 套产品。

如果说前端由于样式的差异较大设计 5 套展现还能接受的话，那么后端明明就是一套业务逻辑(或说差异很小)，为何也要做 5 套呢？我们具体来看 VCom 的这些流程。

- 登录是最基础的服务之一，需要考虑众多产品的快速接入，同时也是个庞大的系统，新产品接入流程通常较烦琐。
- 商品列表和详情页往往是变化较为频繁的页面。
- 购物车和结算由于和最终交易结果挂钩，无论在安全性、可靠性上都是极为严密的，新产品接入也同样烦琐。

同时 VCom 在飞速地成长，新产品的研发试错成本变得更高，这种情况下可预见的是研发速度会被人为地降低（老板们无法容忍），这是研发人员最不愿看到的情况。

好了，VCom 所有流程看下来没有不复杂的，而按照传统思路进行产品的设计又有那么多弊端，为何不从一开始就考虑流程复用的思路呢？

### 5.1.2　示例 2

我们从头再来，VCom 有一套购物主流程：登录、商品列表、商品详情、购物车、结算。

VCom 的产品架构师和技术架构师们很有经验并富有远见。

- 没有急着上第一套 PC 端产品，相反首先为所有流程设计好 API，这些 API 不假定来访问的终端，也就是说可以是 PC、Phone、Pad，无论是 Web 还是 App 都可以访问这一套 API。
- 之后 VCom 首先推出了 Phone Web 产品确保整个 API 的可用性并充分对流程进行适当优化、产出优质的 API 文档。
- 由于整个流程的畅通，Pad Web 版很快也面世了，App 也进入产品规划中，仍然使用同一套 API。

- 之所以选择了 Phone 和 Pad 先下手，VCom 的产品架构师们还没完全考虑清楚产品的核心内容到底是什么，恰好小屏上的有限空间反而让选择变得简单，只有那些对用户有用的内容才需要展现出来。
- 最后顺其自然，PC 版产品终于出来了，核心用的还是那套 API，或许考虑到 PC 的空间较大，会适当地扩展某些 API。

即使忽略穿插其间的“移动优先”理念，示例 2 强大之处还在于一套 API 支持着各个终端，真正实现了流程的复用。流程复用会带来服务端升级成本的上升，但是通过 API 版本化能够很好地得到控制，这部分讨论超出本书的范围，建议有兴趣的读者参考《语义化版本》[1]，如图 5-1 所示。

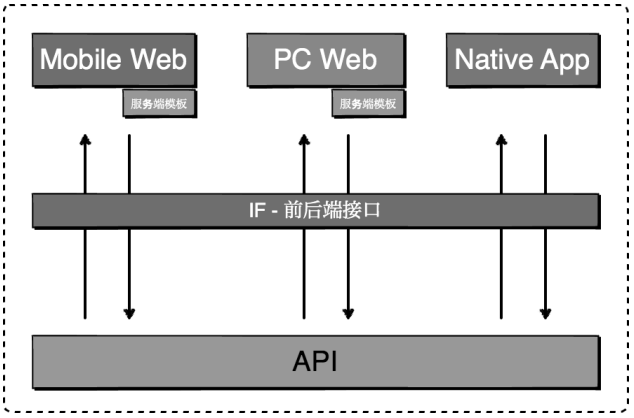



图 5-1 接口实现流程复用

## 5.2 IF

跨终端 Web 需要流程复用，流程复用必然导致前后端解耦开发模式的兴起。接口就成了一个过去人们很少关注，而今又十分重要的存在。笔者在工程实践中遇到过不少接口问题，不断开发工具去解决这些问题，久而久之，工具的积累和对接口认识的不断更新自然促成了 IF（InterFace，取其中 IF 缩写）这个项目。IF 主要包括如下几部分。

- 接口描述：请求、响应数据格式。

[1] http://semver.org/

- 接口文档：由接口描述生成接口文档。
- 接口 Mock：由接口描述生成接口 Mock 数据。
- 接口校验：提供校验服务（HTTP）和校验工具包，支持多种形式的接口校验。

### 5.2.1　始于一次重构

2012 年 9 月，运行了 1 年多的某站点积累了诸多问题。

1. 前后端开发紧密耦合（业务代码甚至出现在 JSP 中）。
2. 易出现前后端相互等待的情况，开发效率低下。
3. 测试同学的工作前松后紧。
4. 随着前后端同学的增多，问题更加严重了。

“进行一次彻底的重构吧！”当时这个念头就冒出来了，后来的数据证明这不仅仅是一次冲动的决定。针对问题 1，前后端必须解耦，否则再难适应后续可预见的业务的爆炸式增长；我们选择了一个合适的 OPOA（One Page One Application，单页应用）框架——ER[2]，前后端完全通过 AJAX 通信，前后端耦合问题减轻了。

随之而来的就是接口问题。

1. 当时前后端习惯了口头约定接口，遇到接口改动往往是前端或后端查阅代码来反向确认接口细节，这在接口较少且变化不大的业务场景或许还能应付。
2. 一旦接口数量爆炸式增长后，且业务初期接口改动又是如此频繁，口头约定已经不能满足业务需求。
3. 于是我们开始人肉维护接口文档，发现了一个很难接受的问题：改动了 Mock Data（调试数据）还需要去修改接口文档，通常是 Mock Data 修改完毕等到即将提交代码时才想起要更新接口文档，这时又得去给 Mock Data 做 diff 后再去改动接口文档。

[2] ER 框架：http://errorrik.com/er/。

本着程序员的“懒人精神”，笔者设计了一种特殊的接口描述文件格式，这是一个能同时运行与 Node 环境与浏览器环境的 js 代码，如代码 5-1 所示。

**代码 5–1 最初的接口描述文件**

```
/**
 * 兼容 Node 和浏览器环境
 */
if (typeof exports === 'undefined') {
    exports = {};
}

exports.config = {
    "name": "这是接口名",
    "desc": "这是接口的详细描述"
};

exports.request = {
    "id": "1000"   // 查询字段
};

exports.response = {
    "success": true, // 标记成功
    "model": {
        "title": "xyz0",
        "list": [// 列表数据
            {// 单条记录
                "id": 1000,
                "name": "name-123"
            },
            {// 单条记录
                "id": 1000,
                "name": "name-123"
            }
        ]
    }
};

exports.responseError = {
    "success": false, // 标记失败
    "model": {
        "error": "Error message"
```

```
    }
};
```

上面的接口描述文件只是一个简化版本，保留了核心部分。

- 已经能看到兼容 Node 和浏览器的逻辑、请求参数列表、响应数据。
- Node 环境下这段代码可以作为一个 node module 加载，笔者用 Node 写了一个文档生成工具，可以把若干份这样的描述文件生成一份接口文档，如图 5-2 所示。
- 同时描述文件又能以 script 形式加载到浏览器环境中供调试用（提取 exports.response）。
- 遇到接口改动，我们只需要修改接口描述文件。


图 5-2　生成的接口文档

项目重构就在修修补补间结束了。在总结这次重构的经验时，笔者又发现了更多的问题。

- 接口文件命名没有规范。
- 接口字段命名没有规范。
- HTTP 请求、响应的编码、Header 等同样没有规范。
- 后端响应的数据可能出现与接口描述文件不一致。
- ……

最初的接口规范被制定出来解决以上问题。之后几次项目中我们又对接口规范做了适当的扩充。

比较有意思的是，最初没有选择诸如 JSON Schema 这样的庞然大物去定义接口。相反，是直接使用 JSON 数据去定义数据格式，听起来不可思议，但从一开始我们就希望“格式定义与调试数据是同一份”，这样可以简化做 Mock 的成本。正是这种适当的简化才让这套机制运转得更加灵活。之后的项目也的确遇到过描述能力不足的问题，比如某些字段可能不存在，这在最初设计中被忽略了，包括对特定字段的格式强校验也被忽略了。不过很快我们又增加了“规则字段”来解决这些问题。

解决了规范问题，却发现测试、后端同学往往没有热情主动校验数据格式是否正确。以上所有的问题笔者都可以通过制定规范、开发工具来解决，现在遇到的是一个流程问题，是需要说服测试或后端增加额外的工作量才能做到的一件事情。在解决了最后的这个障碍后，一个完整的接口工作流程如图 5-3 所示。

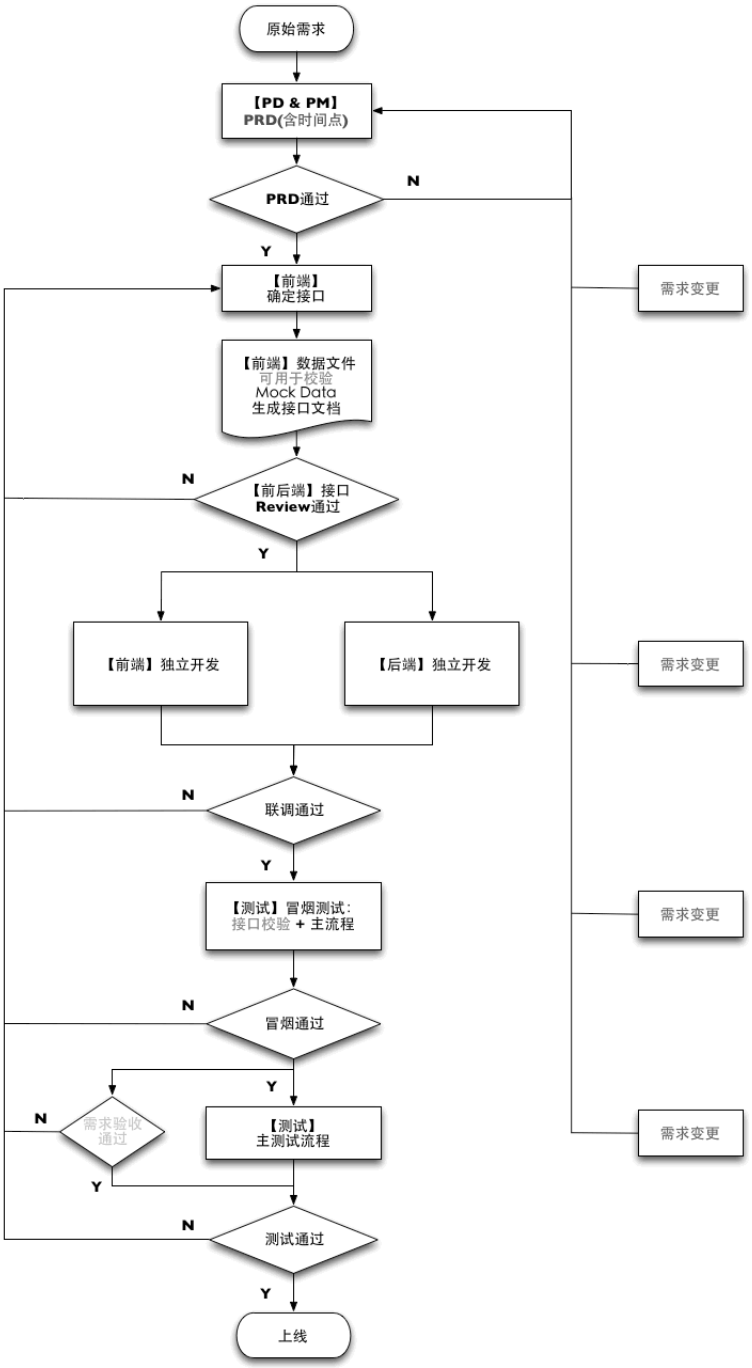

原始需求
【PD & PM】
PRD(含时间点)
PRD通过
N
Y
【前端】
确定接口
需求变更
【前端】数据文件
可用于校验
Mock Data
生成接口文档
N
【前后端】接口
Review通过
Y
【前端】独立开发
【后端】独立开发
需求变更
N
联调通过
Y
【测试】冒烟测试：
接口校验 + 主流程
需求变更
N
冒烟通过
Y
N
需求验收
通过
【测试】
主测试流程
需求变更
Y
N
测试通过
Y
上线

图 5-3　接口工作流程

### 5.2.2 新的环境

时间来到了 2013 年 4 月，笔者从百度来到了天猫。接手的第一个项目是一个面向商家的工具：复杂业务逻辑导致了复杂的表单交互，包含大量需要局部刷新的需求。显然 OPOA 成了首选，之后便是制定接口、生成接口文档、接口 Mock、接口校验。一件事情重复去做的时候也是提取共性、抽象出本质的绝佳时机，在和其他工程师做了几次邮件交流后，我们决心把接口作为一个独立的项目来推进，至此 IF（InterFace）诞生了。

笔者在百度时所处的是一个小型垂直团队（50 人以内，有前端、后端、测试、产品等角色），无论接口定义还是接口校验推动相对迅速。新的环境则完全不同，所有角色都是独立的团队并且已经有一套较为成熟的开发模式。摆在眼前的问题变成了：接口定义和接口校验该由谁来执行？

在已有的工作模式下无法解答这个问题，我们只能从生活中寻求模型。

### 5.2.3 模型

先看看市场中的场景，即 IF 模型，如图 5-4 所示。

- 生产者生成什么消费者再去消费，这种模式已经成为过去时。
- 正式消费者的需求推动了某个产品的诞生、改进、消亡。
- 产品中涉及人身安全的基本规范会由政府机构给出，并由另一部门机构监管。来到接口的场景下，前端不正是接口的消费者吗？
- 前端提出对接口的需求，规定好格式，也只有前端最清楚自己需要什么样的数据，以及数据可能的扩展方向。相反如果由后端确定数据格式，往往返回的是一个极其接近数据库存储结构的数据结构，但是这样的结构往往不是前端（消费者）需要的。
- 后端作为接口的制造者，首先要做的就是满足数据定义，为了确保格式能够满足，应该增加接口的单元测试。
- 测试在其冒烟测试阶段应该检测接口单元测试的结果，这种检查包括单测的覆盖率以及通过率。

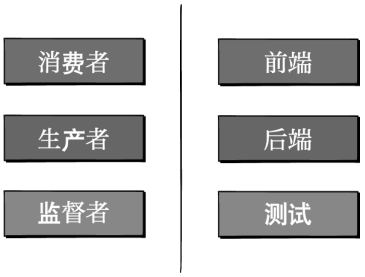



图 5-4　IF 模型

正是这个模型解决了前面的问题“接口定义和接口校验该由谁来执行？”

- 接口定义由接口的需求方来制定，绝大部分业务场景下会是前端。
- 接口校验由接口的生产者来执行，绝大部分业务场景下会是后端。
- 测试作为监督者需要在上线流程中确保接口校验的质量，同时也非常推荐测试提供自动化工具参与接口校验。

### 5.2.4　解决方案

有了模型后，IF 的解决方案更加清晰了。

1. 规范：统一的接口规范。
2. 文档：对所有角色有约束的接口文档。
3. 校验：数据校验工具（UI 工具和 HTTP 服务）、后端数据 UT。
4. 流程：接口改动通知、接口测试保障。

从 IF 解决方案的角度来看，笔者接手的项目存在如下问题。

1. 规范：没有统一的接口规范。
2. 文档：前后端口头约定（或查代码）。
3. 校验：后端缺少接口数据校验。
4. 流程：接口改动无通知，无测试保障，可能（真实发生过）出现线上接口故障。

### 5.2.5　架构

有了解决方案，下一步便是重新整理 IF 的架构，IF 架构如图 5-5 所示。

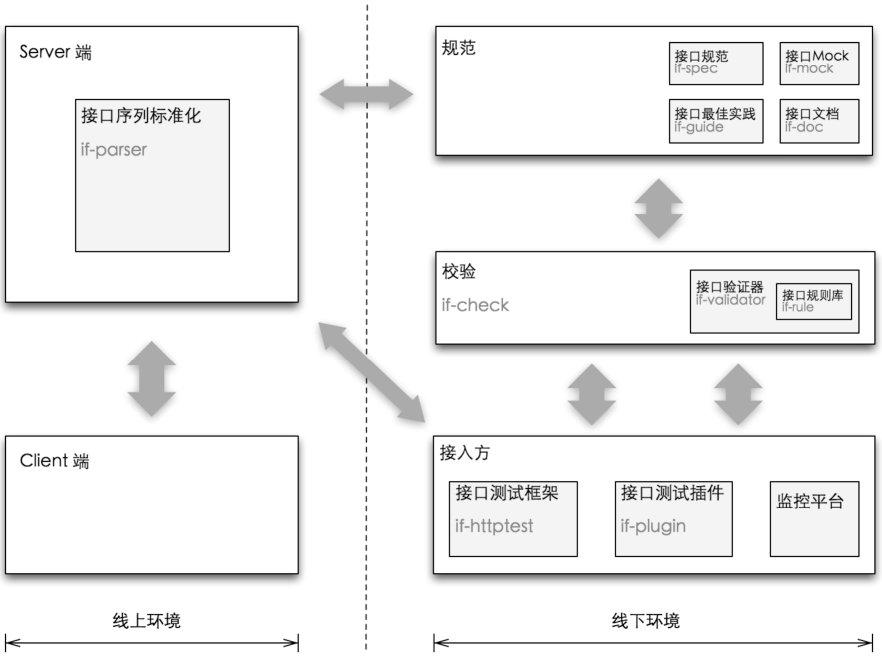



图 5-5 IF 架构

几个重要模块如下。

- if-spec，接口规范，负责对每个接口的请求/响应（支持按条件定义响应）的数据结构的定义。一份最简单的 if-spec 文件如“代码 5-2”所示，其中 request、response 数据格式遵守 IETF JSON Schema Draft-04[3]。
- if-doc，支持从 if-spec 生成接口文档。
- if-mock，支持从 if-spec 生成 Mock 数据（用于调试），充分利用 JSON Schema 对数据的描述能力。
- if-guide 接口最佳实践。
- if-check 接口校验，负责对响应数据及对应的 if-spec 文件进行校验。if-check 提供了：HTTP 校验服务、UI 校验工具，均调用了 if-validator。包含的子模块有两个。
  - if-validator，执行接口校验的工具，被 if-check 调用，已有大量的开源工

[3] http://json-schema.org/documentation.html

具可选[4]。

- if-rule，校验规则库，通过扩展 JSON Schema 的“format”属性实现，可以实现持续积累通用规则。

- 接入方如下。
  - if-httptest 接口测试框架，面向后端的接口单元测试、自动化接口测试框架，提供模拟登录等功能，调用 if-check 的 HTTP 校验服务。
  - if-plugin 接口测试插件，以浏览器插件形式提供，方便在页面中集成接口校验。
  - 监控平台，内部的质量监控系统。

- if-parser 接口序列化标准化工具，通过扩展 JSON Schema 的 Core 部分，增加跨语言转换的规范，并用扩展后的 schema 规范配合 if-spec 将服务端对象转换为 JSON 数据。如代码 5-2 所示为 if-spec 示例。

**代码 5–2　if–spec 示例**

```
{
    "meta": {
        "name": "basic"
    },
    "request": {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "type": "object"
    },
    "response": {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "type": "object",
        "properties": {
            "status": {
                "type": "integer"
            }
        }
    }
}
```

最后来看一下 IF 所有模块在工作流程中的位置，如图 5-6 所示。

[4] http://json-schema.org/implementations.html

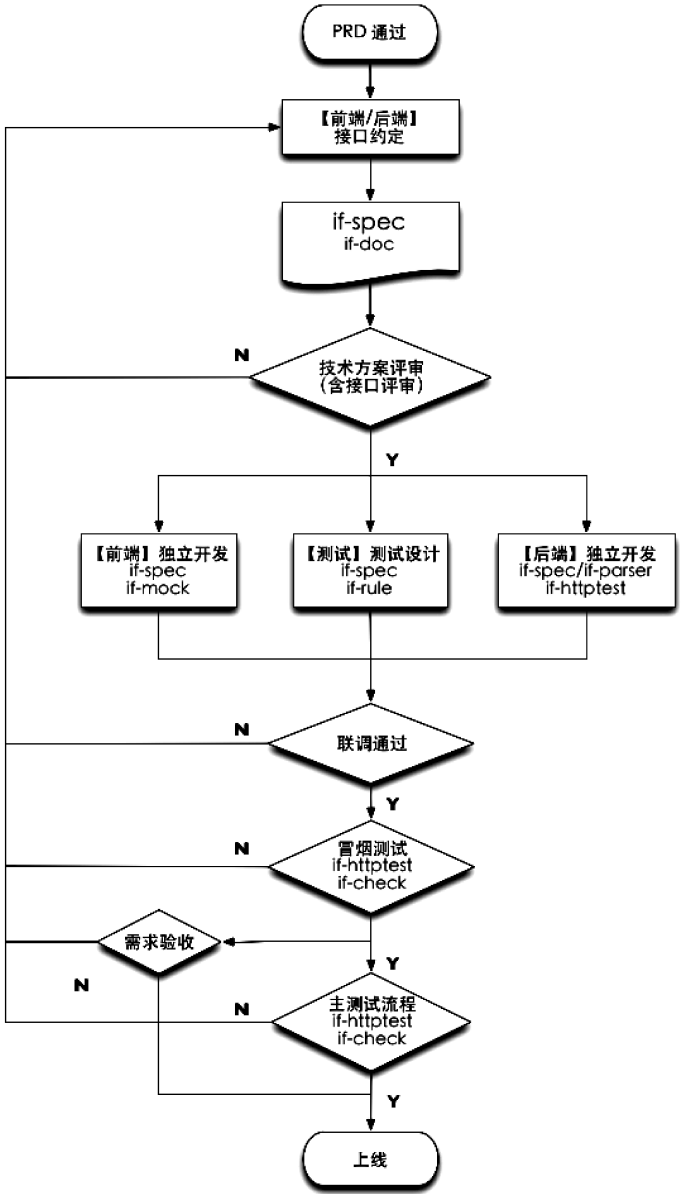



图 5-6 IF 融入开发流程

## 5.2.6 路线图

回顾 IF 产生的最初环境和目前所处的工程环境，笔者列出了实现 IF 解决方案和架构的路线图（见图 5-7）。

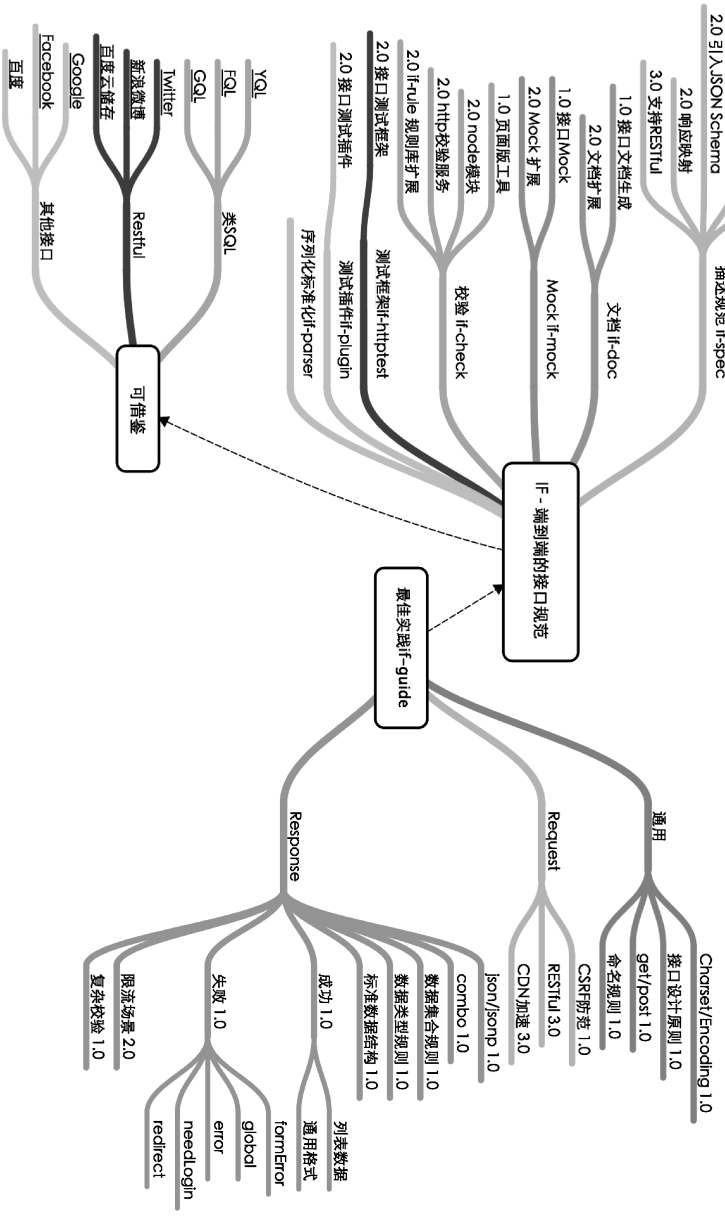



图 5-7　IF 路线图

- 1.0
  - 规范：if-spec 接口规范 1.0，if-guide 1.0。
  - 文档：if-doc 1.0，通过 if-spec 生成接口文档。

- Mock：if-mock 1.0，基本的 Mock 功能。
- 校验：if-check 1.0 接口校验服务搭建完成。
- 流程：给出推荐的工作流程并在个别项目试点。

- 2.0
  - 规范：if-spec 2.0 引入 JSON Schema 作为数据描述格式，if-guide 2.0。
  - 文档：if-doc 2.0 富交互的接口文档。
  - Mock：if-mock 2.0，更完善的 Mock，通过 if-spec Mock 接口，充分利用了 JSON Schema 对接口的描述能力。
  - 校验：if-check 2.0 接口校验服务、if-httptest 接口测试框架、if-plugin 接口测试插件。
  - 流程：更大范围推广优化后的工作流程。

IF 1.0 已经完成，读者在安装了 Node.js[5]的环境下执行以下命令安装 ift：

```
npm install ift
```

Mac 用户可能需要执行：

```
sudo npm install ift
```

之后可以使用 ift 的文档生成、接口 Mock 功能，以及调用 if-check 的 node 校验模块：

1. 运行“ift -i”初始化接口环境。
2. 运行“ift -s”生成接口文档。
3. 运行“ift -e”启动接口 Mock 服务器。
4. 使用 node 代码“require（‘ift’）.ifCheck（{format:{}}, {data:{}}）”进行接口校验。

“README.mk”中有以上命令的详细介绍。

截止本书初稿完成时（2014.2），2.0 正在开发过程中，更多精彩的特性被引入到 2.0 中。

1. if-spec 中的请求和响应数据格式使用了 JSON Schema，具有更强大的接口描述能力。
2. 1.0 的接口文档是基于 markdown 的静态页面，2.0 中会生成更富有交互性

[5] http://nodejs.org/

且信息更清晰的接口文档。

3. if-httptest 是与测试同学合作的子项目，目前已经开始试用，会为后端的接口单元测试提供更多便利性。
4. if-plugin 也是与测试同学合作的子项目，即将启动开发，方便在浏览器中快速进行接口校验。
5. if-parser 是与后端同学合作的子项目，着重实现后端 JSON 数据序列化的标准化。

IF 2.0 中测试和后端的加入需要一种通用的、扩展性良好的数据描述格式，因而选择 JSON Schema 作为 if-spec 的数据描述格式也成了必然的选择。

## 5.3 if-spec 2.0

if-spec 2.0 是数据接口规范，包含：

- meta，接口元信息。
- request，请求数据的 schema。
- response，默认响应数据的 schema。

在继续之前，请先了解 JSON Schema 的基本格式。

### 5.3.1 JSON Schema

if-spec 2.0 使用 JSON Schema 作为数据描述格式，本节详细讨论 JSON Schema。

JSON Schema 是 Gary Court 向 IETF 提交的用于描述 JSON 数据结构的规范，目前版本是 Draft-V4[6]（简称为 V4），V4 相对此前的 V0～V3[7]做了许多有益的扩展[8]。尽管 V4 还未成为正式协议，就目前的发展势头来说，已然有成为行业标准的势头。笔者曾在自定义标准和 V4 之间挣扎，最终从 IF 协作开发的现实需要和长远发展角度考虑还是选择了 V4 作为接口数据描述的协议。

---

6 JSON Schema 最新版：http://json-schema.org/documentation.html。

7 JSON Schema V3：http://tools.ietf.org/html/draft-zyp-json-schema-03，其他版本均可在 IETF 搜索到。

8 V4 ChangeLog：http://json-schema.org/latest/json-schema-validation.html#anchor145。

#### 5.3.1.1 示例

从 JSON Schema 的官方网站[9]上能看到所有关于 JSON Schema 的信息，包括协议原本、Demo、配套工具集等，在详细介绍 JSON Schema 之前，我们通过几个简单的 Demo 来了解下 JSON Schema 是如何描述 JSON 数据的，如代码 5-3 和代码 5-4 所示。

**代码 5–3 JSON 数据**

```
{
   "firstName": "luics",
 "lastName": "xu",
 "age": 25
}
```

**代码 5–4 上一数据对应的 JSON Schema**

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "title": "Example Schema",
 "description": "Example Schema 描述信息",
 "type": "object",
 "properties": {
     "firstName": {
         "type": "string"
     },
     "lastName": {
         "type": "string"
     },
     "age": {
         "description": "Age in years",
         "type": "integer",
         "minimum": 0
     }
 },
 "required": ["firstName", "lastName"]
}
```

上面的示例涵盖了 JSON Schema 最核心的几个要素。

1. JSON Schema 使用 JSON 格式书写，简称 schema，所描述的数据通常称为

[9] http://json-schema.org/

实例（instance）。

2. 第一个属性（Property）“$schema”是推荐使用的，用于标识 JSON Schema 版本号，这个标识会用在一些校验工具中。
3. 根路径下的“title”、“description”分别描述 Schema 的短标题和详细信息。
4. 根路径下的“type”、“properties”、“required”分别描述 instance 最外层结构的数据类型、属性集合和必选属性集合。
5. 根路径下的“properties”中有“firstName”、“lastName”、“age”3 个属性，每个属性的值又是一个 schema。
6. “age”下的属性“type”和“minimum”指定“age”的值必须是大于等于 0 的整数。

下面的几个 instance 均不满足上述 schema。"

1. 缺少了“lastName”，不满足“"required": ["firstName", "lastName"]”

```
{
 "firstName": "luics",
 "age": 25
}
```

2. “firstName”，不满足“"firstName": {"type": "string"}”

```
{
 "firstName": 37,
 "lastName": "xu",
 "age": 25
}
```

3. “age”，不满足“"age": {"minimum": 0}”

```
{
 "firstName": "luics",
 "lastName": "xu",
 "age": -1
}
```

V4 包括以下 3 个部分。

1. Core：定义了 JSON Schema 的基本要素。
2. Validation：定义了 JSON Schema 验证关键字及校验算法。
3. Hyper-Schema：定义了 JSON Schema 的超媒体表示方法。

接下来的章节将会详细介绍 Core 和 Validation。

#### 5.3.1.2 Core

本节给出 JSON Schema Core 中和 IF 紧密相关的内容，完整译本请参见书末“附录 B JSON Schema Core”。区别于本书其他章节序号，以下序号以 C 为前缀。

### C.1. 引言

JSON Schema 是用于定义 JSON 数据结构的 JSON 媒体类型（media type）。 JSON Schema 为特定应用程序提供了 JSON 数据的约定并规范了如何与之交互。JSON Schema 旨在为 JSON 数据定义校验、文档和超链接导航以及交互控制。

### C.2. 核心术语

#### C.2.1 property、item

当提及 JSON Object（定义在 RFC 4627），术语“member”（成员）和“property”（属性）是可以互换使用的。

当提及 JSON Array（定义在 RFC 4627），术语“element”（元素）和“item”（成员）是可以互换使用的。

#### C.2.2 JSON Schema、keywords

JSON Schema 是一个 JSON 文档，文档的内容必须（MUST）是一个对象。由 JSON Schema（本规范或相关规范）所定义的对象成员被称为关键字（keywords）或 schema 关键字。JSON Schema 可以（MAY）包含 schema 关键字以外的属性。

#### C.2.3 Empty schema

一个没有任何属性或属性均为非 schema 关键字的 JSON Schema 被称为空（Empty）Schema。

#### C.2.4 Root schema、subschema

下面这个 JSON Schema 没有 subschema（子 schema）。

```
{
  "title": "root"
}
```

JSON Schema 可以嵌套，如：

```
{
    "title": "root",
    "otherSchema": {
        "title": "nested",
        "anotherSchema": {
            "title": "alsoNested"
        }
    }
}
```

上面例子中的“nested”和“alsoNested”是 subschema，“root”是 root schema。

### C.2.5　JSON Schema 原生类型

JSON Schema 为 JSON 值定义了 7 种 primitive（原生）类型。

- array：JSON array。
- boolean：JSON boolean。
- integer：JSON number，不包含小数和指数部分。
- number：JSON number，该类型包括了“integer”。
- null：JSON null。
- object：JSON object。
- string：JSON string。

### C.2.6　JSON 值判等

两个 JSON 值相等当且仅当：

- 均为 null；或
- 均为 boolean，且有相同的值；或
- 均为 string，且有相同的值；或
- 均为 number，且有相同的算术值；或
- 均为 array，且
  - 相同的数组成员数，且
  - 相同索引下的元素的值相等，或
- 均为 object，且
  - 属性名集合相同，且
  - 相同属性名下的元素的值相等。

### C.2.7 Instance

Instance(实例)是任何 JSON 值。一个 instance 可以由一个或多个 schema 描述。instance 也被称为“JSON instance”或“JSON data”。

## 5.3.1.3 Validation

本节整理了一个 JSON Schema Validaton（以下简称 Validaton）的关键字列表（见表 5-1 和表 5-2），完整译文请参见书末“附录 C JSON Schema Validaton”。Validaton 是 JSON Schema 中用于定义校验关键字的规范，这些关键字正是 JSON Schema 文档中最主要的组成部分。

表 5-1 JSON Schema Validation 关键字

| 适用类型 | 关键字 | 合法值 | 默认值 | 含义及验证通过条件 |
|---|---|---|---|---|
| | title | string | - | schema 短标题 |
| | description | string | - | schema 长描述 |
| | default | string | - | 指定默认值 |
| | format | string | - | 语义化校验，详见下文 |
| - | | | | |
| 所有 | enum | array 至少 1 元素，元素唯一 | - | 枚举 |
| 所有 | type | string 或 array<br>7 种原生类型 | - | 数据类型 |
| 所有 | allOf | array 至少 1 元素，元素为 schema | - | 通过所有验证 |
| 所有 | anyOf | array 至少 1 元素，<br>元素为 schema | - | 通过任一验证 |
| 所有 | oneOf | array 至少 1 元素，元素为 schema | - | 只通过一个验证 |
| 所有 | not | schema | - | 不通过该验证 |
| 所有 | definitions | object<br>属性为 schema | | |
| - | | | | |
| number | multipleOf | integer 且大于 0 | - | 整数可被该值整除 |
| number | maximum | number | - | 数字的最大值 |

续表

| 适用类型 | 关键字 | 合法值 | 默认值 | 含义及验证通过条件 |
|---|---|---|---|---|
| number | exclusiveMaximum | boolean | false | 不含最大值 |
| number | minimum | number | - | 数字的最小值 |
| number | exclusiveMinimum | boolean | false | 不含最小值 |
| string | maxLength | integer 且大于等于 0 | - | 字符串最大长度 |
| string | minLength | integer 且大于等于 0 | 0 | 字符串最小长度 |
| string | pattern | 正则表达式 | - | 字符串正则匹配 |
| array | items | schema 或 array | 空 schema | 数组元素定义 |
| array | additionalItems | boolean 或 schema | 空 schema | 额外数组元素定义 |
| array | maxItems | integer 且大于等于 0 | - | 数组最大元素个数 |
| array | minItems | integer 且大于等于 0 | 0 | 数组最小元素个数 |
| array | uniqueItems | boolean | false | 数组所有元素唯一 |
| object | maxProperties | integer 且大于等于 0 | - | 对象最大属性个数 |
| object | minProperties | integer 且大于等于 0 | 0 | 对象最小属性个数 |
| object | required | array 至少 1 元素，元素为 string 且唯一 | - | 必须出现的属性集 |
| object | additionalProperties | boolean 或 schema | {} | 额外属性定义 |
| object | properties | schema | {} | 属性定义 |
| object | patternProperties | schema | {} | 属性名为正则的属性定义 |
| object | dependencies | schema | - | 分为 schema 依赖和属性依赖 |

表 5-2　Validation format 属性

| 适用类型 | 属性 | 定义 | 含义 |
|---|---|---|---|
| string | date-time | RFC3339 section 5.6 | 日期时间格式 |
| string | email | RFC5322 section 3.4.1 | E-mail 地址格式 |
| string | hostname | RFC1034 section 3.1 | hostname 格式 |
| string | ipv4 | RFC2673 section 3.2 | IPv4 格式 |
| string | ipv6 | RFC2373 section 2.2 | IPv6 格式 |
| string | uri | RFC3986 | URI 格式 |

## 5.3.2 Demo

下面分别展示了 if-spec 简单、基本和高级的版本。

### 5.3.2.1 简单 Demo

这个 if-spec 描述了如下几点，简单 Demo 如代码 5-5 所示。

1. meta、request、response 三个字段是必选字段（required）。
2. meta.name 是必选字段，定义了接口名为“empty”。
3. request、response 可以是“空 Schema”。

**代码 5–5 简单 Demo**

```
{
    "meta": {
        "name": "empty"
    },
    "request": {},
    "response": {}
}
```

### 5.3.2.2 基本 Demo

这个 if-spec 描述了如下几点，基本 Demo 如代码 5-6 所示。

1. meta.name 是必选字段，定义了接口名为“basic”。
2. request，定义了请求参数（URL QueryString）中 search 类型为 string，且 search 是必选的（required）。
3. response，定义了响应数据中必选字段 status 的类型为 integer。

**代码 5–6 基本 Demo**

```
{
    "meta": {
        "name": "basic"
    },
    "request": {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "type": "object",
        "required": ["search"],
        "properties": {
```

```
            "search": {
                "type": "string"
            }
        }
    },
    "response": {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "type": "object",
        "required": ["status"],
        "properties": {
            "status": {
                "type": "integer"
            }
        }
    }
}
```

#### 5.3.2.3　高级 Demo

这个 Demo（如代码 5-7 所示）较完整地展现了 if-spec 的特性，描述了：

1. meta.name，定义了接口名为“basic”，meta.description 是接口描述信息，meta.type 指定接口支持的 HTTP Method 类型。
2. request，定义了请求参数（URL QueryString）中 search 类型为 string，且 search 是必选的（required），type（数据类型）为 string，且 type 不是必选的（not required）。
3. response、response2、responseError 是 3 个 response schema。
4. meta.responseMap，定义了 response schema 的映射关系：
    - 元素 1 定义了规则：响应数据 status 为−1 时，response schema 为 responseError。
    - 元素 2 定义了规则：请求参数 search 的值满足正则表达式 /[A-Z]/，且响应参数 status 为 1 时，数据 response schema 为 response2。
    - 遍历映射关系，遇到满足条件则退出。

**代码 5–7　高级 Demo**

```
{
    "meta": {
```

```
    "name": "这是接口名",
    "description": "这是接口的详细描述",
    "type": ["GET", "POST"],
    "responseMap": [
        {
            "rule": [
                {
                    "type": "response",
                    "property": "status",
                    "value": "-1"
                }
            ],
            "schema": "responseError"
        },
        {
            "rule": [
                {
                    "type": "request",
                    "property": "search",
                    "pattern": "[A-Z]"
                },
                {
                    "type": "response",
                    "property": "status",
                    "value": "1"
                }
            ],
            "schema": "response2"
        }
    ]
},
"request": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "required": ["search"],
    "properties": {
        "search": {
            "type": "string",
            "description": "搜索关键字"
        },
        "type": {
            "type": "string",
```

```
            "description": "区分 mobile、pc 接口"
        }
    }
},
"response": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "required": ["status"],
    "properties": {
        "status": {
            "type": "integer",
            "description": "状态码"
        },
        "data": {
            "type": "object",
            "description": "服务端返回的正常数据",
            "properties": {
                "items": {
                    "type": "array",
                    "description": "搜索返回结果列表",
                    "items": {
                        "itemId": {
                            "type": "string",
                            "description": "商品 ID"
                        }
                    }
                },
                "key": {
                    "type": "string",
                    "description": "搜索关键字"
                }
            }
        }
    }
},
"response2": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "required": ["status"],
    "properties": {
        "status": {
            "type": "integer"
```

```
            }
        }
    },
    "responseError": {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "type": "object",
        "required": ["status"],
        "properties": {
            "status": {
                "type": "integer"
            }
        }
    }
}
```

### 5.3.3 meta

“附件 - if-spec 2.0”是描述 if-spec 2.0 的 JSON Schema。

meta 保存接口配置，必须（MUST）是 JSON Object。

meta.responseMap 用于映射响应 Schema，补足 JSON Schema 在“条件描述”上的缺陷。

meta.responseMap 的每个成员是一个映射规则集合，目前支持重定向类型（type），每个 type 有对应的一组关键词。

- request，根据请求参数重定向。
  - property：URL 参数名称
  - data：POST 数据项名称
  - value：普通值
  - pattern：正则表达式，不区分大小写
- response，根据响应数据重定向。
  - property：成员名称
  - value：普通值
  - pattern：正则表达式，不区分大小写
- method，根据 HTTP Method 重定向，这是为 RESTful 预留的扩展。
  - method：HTTP Method

#### responseMap 参考算法

以下算法供接口校验时作为参考。

1. meta.responseMap 是一个数组记为 A，A 的成员记为 B。
2. B.rule 是一个规则集合，规则间是“与”关系。
3. 遍历 A，遇到 B.rule 匹配成功时停止遍历，B.schema 指向的即是响应数据格式。
4. 如果遍历结束仍未有匹配，则使用 response 作为响应数据格式。

### 5.3.4　if-spec 1.0

#### 5.3.4.1　请求

使用如下格式描述一个典型的请求。

1. 第 1 行表示请求标识。
2. 第 2 行开始以 JSON 描述请求参数，注释可以提供更多描述。
   - 单行注释应该（SHOULD）置于行末。
   - 多行注释应该（SHOULD）置于行前。下面是以 HTTP POST 请求资源 /exampleUrl 携带参数 page、name 的文档。

```
POST /exampleUrl
{
    "page": 1,        // 页码，首页为 1
    "name": ""        // 其他查询字段
}
```

#### 5.3.4.2　响应

使用如下格式描述一个典型的响应。

1. 使用 JSON 描述响应数据，注释规则同请求文档。
2. 后缀__RULE__代表规则字段（参见后续“规则字段”）。

```
{
    "success": true, // 标记成功
    "model": {
```

```
        // 规则：或者字段「title」不存在，若存在则「title」的值必须
        // 匹配正则表达式（不区分大小写，以 xyz 开头，数字结尾）
        "title__RULE__": "{{not-required}},{{/^xyz\\d+$/i}}",
        "title": "xyz0",        // 规则：字段「list」允许不存在
        "list__RULE__": "{{not-required}}",
        "list": [// 列表数据
            {// 单条记录
                "id": 1000,
                "name": "name-123"
            }
        ]
    }
}
```

#### 5.3.4.3 键集合

1. 响应数据的键集合（Key Set）必须（MUST）是格式文档中定义的子集。
2. 键集合严格等于约定的键集合是可选的（OPTIONAL）。

#### 5.3.4.4 值类型

1. 值类型必须（MUST）和接口文档定义保持一致。
   - 如约定了{"filed": false}，不可返回 {"filed": "false"}，布尔判断中这两个结果恰好相反。
2. 空值时仍必须（MUST）满足上一条。
   - 对象 {"filed": {}}
   - 数组 {"filed": []}
   - 字符 {"filed": ""}
   - 数字 {"filed": 0}
   - 布尔 {"filed": false}

#### 5.3.4.5 规则字段

引入规则字段强化校验功能，且规则字段的约束可以覆盖键集合和值类型的规则。

1. 规则字段的名称均包含后缀__RULE__。如 name__RULE__是一个规则字段，为字段 name 指定规则。

2. 目前规定了两类规则，分别如下。
   - {{not-required}}：该规则允许指定字段不存在，字段默认需存在。
   - {{/regexp string/i}}：指定字段的值必须（MUST）匹配正则表达式[10]。
3. 规则可叠加，以英文逗号分割。
   - "name__RULE__": "{{not-required}},{{/^xyz\\d+$/i}}" 上面规则为：或者字段`name`不存在，若存在则`name`的值必须匹配正则表达式（不区分大小写，以 xyz 开头，数字结尾）。

## 5.4　if-mock 2.0

if-mock 是接口 Mock 工具，充分利用了 JSON Schema 的数据描述能力，将 if-spec 转换为 Mock 数据。

“代码 5-8”这份 Mock 数据是由“代码 5-9”通过 if-mock 生成的。

1. items 具有约束“*"minItems": 1*”、“*"maxItems": 6*”，因而 Mock 数据中 items 数据成员个数最少 1 个，最多 6 个。
2. itemId 的 Mock 数据在 1～999999 之间。
3. title 的 Mock 数据长度在 18～26 之间。
4. test 字段不在“required”定义中，所以 Mock 数据中 items 第一个元素就没有 test 字段；同时 test 字段需要满足“enum:[-1,0,1,2]”，因而 Mock 数据中只会出现这几个固定值。

**代码 5–8　Mock 数据**

```
{
    "items": [
        {
            "itemId": 209535,
            "title": "*WOvNz*pfxZ%35s6wFU"
        },
        {
            "itemId": 957994,
```

[10] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp

```
            "title": "dJVqztChtWTfL7^hUS",
            "test": 1
        }
    ]
}
```

**代码 5–9 用于 Mock 的 if–spec 文件**

```
{
    "meta": {
        "name": "Mock Demo"
    },
    "request": {},
    "response": {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "type": "object",
        "properties": {
            "items": {
                "type": "array",
                "description": "搜索返回结果列表",
                "maxItems": 6,
                "minItems": 1,
                "items": {
                    "type": "object",
                    "properties": {
                        "itemId": {
                            "type": "integer",
                            "minimum": 1,
                            "maximum": 999999
                        },
                        "title": {
                            "type": "string",
                            "maxLength": 26,
                            "minLength": 18
                        },
                        "test": {
                            "type": "number",
                            "enum": [-1, 0, 1, 2]
                        }
                    },
                    "required": [
                        "itemId",
                        "title"
```

```
                    ]
                }
            }
        }
    }
}
```

### if-mock 1.0

if-mock 1.0 直接返回 if-spec 1.0 文件中的 exports.response 部分，未做更多的控制。

## 5.5　if-guide 2.0

if-guide 是接口最佳实践，为表述上更加严谨，这里沿用了 RFC2119 中的相关术语。区别于本书其他章节序号，以下序号以 S 为前缀。

### S 1.　简介

IF 是端到端的接口规范。本文档是端到端通信的双方需要遵守的规范。

### S 2.　要求

在本文档中，使用的关键字会以中文+括号包含的关键字英文表示：必须（MUST）。关键字"MUST"、"MUST NOT"、"REQUIRED"、"SHALL"、"SHALL NOT"、"SHOULD"、"SHOULD NOT"、"RECOMMENDED"、"MAY 和“OPTIONAL"”被定义在 rfc2119 中。

常用关键字：

- 必须（MUST），强制执行，不得违反。
- 应该（SHOULD），强烈推荐执行，无特殊情况不得违反。
- 可以（MAY），仅供参考，根据具体情况确定方案。

### S 3.　if-spec

参见本章“if-spec 2.0”章节。

### S 4.　接口模型

引入现实世界中的“消费者—生产者—监管者”模型。

1. 消费者提出需求，生产者满足消费者需求生产产品，监管者确保产品的基本品质。
2. 格式文档的发起者好比消费者，向处于生产者地位的接口数据提供方提出接口需求，而接口校验保证了响应数据满足格式文档。
3. 接口定义应该（SHOULD）由接口需求方来制定，绝大部分业务场景下会是前端。
4. 接口校验应该（SHOULD）由接口生产者来制定，绝大部分业务场景下会是后端。
5. 测试作为监督者应该（SHOULD）在上线流程中确保接口校验的质量。

## S 5. HTTP 请求

本部分针对通信双方使用 HTTP 协议的场景。

### S 5.1. 编码

1. 请求参数应该（SHOULD）使用 UTF-8 编码，必须（MUST）做 URL encoding。
2. 某些后台系统默认编码是 GBK，可以（MAY）在 URL 中使用参数告知请求编码，如某系统中使用__charset=UTF-8。下面展示了满足以上要求的一个例子，其中 p2 是中文“测试”：

```
http://www.example.com?__charset=UTF-8&p1=a%20b&p2=%E6%B5%8B%E8%AF%95
```

### S 5.2. HTTP METHOD

AJAX 中使用的 HTTP METHOD，根据应用场景选择。

1. 前台应用场景，应该（SHOULD）优先考虑 GET，URL 长度可能超过 1KB 时必须（MUST）使用 POST。
2. 后台应用场景，应该（SHOULD）考虑统一使用 POST。
3. GET、POST 具有不同的语义[11]，GET 应该（SHOULD）只用于查询信息。
4. PUT、DELETE 具有更丰富的语意，浏览器兼容性没有问题，可以（MAY）尝试。

[11] http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

### S 5.3. 格式

请求参数必须（MUST）满足 if-spec 规范。

## S 6. HTTP 响应

本部分针对通信双方使用 HTTP 协议的场景。

### S 6.1. 响应数据

1. 格式应该（SHOULD）为 JSON[12]或 JSONP[13]，可以（MAY）通过 callback 参数切换。
2. 可以（MAY）同时支持 POST 和 GET。
3. 可以（MAY）支持 PUT 和 DELETE。

### S 6.2. JSONP

响应数据格式为 JSONP 时，可以（MAY）使用“window.jsonpxxx &&”前缀，避免访问超时情况下客户端产生脚本报错：

```
window.jsonpxxx && jsonpxxx({....})
```

### S 6.3. 跳转

出现 302 跳转时，最终的响应数据应该（SHOULD）仍然满足约定的数据格式，不应该（SHOULD NOT）跳转至一个 HTML 页面（如出错页）。

### S 6.4. 编码

1. 响应数据为 JSON 时，必须（MUST）设置 HTTP HEADER，charset 必须（MUST）指定编码，防止中文出现乱码，如：
   ```
   Content-Type:application/json;charset=gbk
   ```
2. 响应数据为 jsonp 时，必须（MUST）设置 script 标签的 charset 属性；如果无法设置 charset，必须（MUST）将数据中的中文做 ASCII 化处理。以上两个处理均是防止中文在某些浏览器（已知 IE 6）下乱码：
   ```
   <script charset="utf-8" src="..." />
   ```

### S 6.5. 格式

响应数据必须（MUST）满足 if-spec 规范。

---

[12] http://www.json.org/

[13] http://json-p.org/

## S 7. 推荐实例

### S 7.1. 成功时的通用格式

```
{
    "success" : true,
    "model" : {
        "field1" : "value1",//建议增加版本号说明，如下的“[1.6]”代表
filed2 字段在版本 1.6 新增或改动
        "field2" : "value2",//[1.6] field2 详细说明
        "totalSize" : 5,      //总记录数
        //列表数据；key 固定为 list，支持多列表
        "list" : [
            {//一条记录
                "id": 1,
                "name": "abcd"
            },
            {
                "id": 2,
                "name": "abcd"
            }
        ]
    }
}
```

表单提交成功时通常 model 为 {}，如需传递信息亦可，如下：

```
{
  "success" : true,
  "model" : {
      "someMsg" : "value1",
      "otherMsg" : "value2"
  }
}
```

### S 7.2. 表单提交失败

```
{
    "success" : false,
    "model" : {
        // formError 是特殊的 key!
        // 前端通过 check 该 key 确定是否表单错误
        "formError":{
            "field1" : "error1",
```

```
            "field2" : "error2"
        }
    }
}
```

**S 7.3.　其他失败情况**

```
{
    "success": false
    "model": {
        "{type}" : {value} // {type} 详见表 5-3
    }
}
```

表 5–3　{type} 的取值

| {type} | {value} 典型值 | 备注 |
|---|---|---|
| global | "global error message" | 全局性错误（如后端服务异常），前端会使用警告对话框显示出错信息，或跳转至出错页 |
| error | "error message" | 业务级错误（非表单），常见于 AJAX 响应 |
| needLogin | true | 需要登录时返回该字段，通常是未登录或会话超时导致的登录 fviu 的失效 |
| redirect | "redirect url" | 重定向 |

## 5.6　总结

笔者曾为 IF 设定过几条成功的标准。

1. 让更多工程师意识到接口的存在性和重要性。
2. 让更多工程师能以 IF 的设计思路作为处理接口问题的基点。
3. 找到 IF 的 Scope：做什么和不做什么。

第 1 条是最重要的。

第 2 条是更加理想的状态，接口涉及几乎所有的工程师，IF 本身并不限定是由前端、后端或者测试来推进的；同样更不限定工程师们用更加优雅的方式处理接口定义、文档、校验、Mock 等工作，笔者只希望 IF 的设计思路（模型、解决方案、架构）能够有幸成为这些优雅方式的基点。

而第 3 条更像是笔者对自己提的要求，直到此刻：

① 仍然不确定是否需要一个 UI 界面来简化接口的定义。

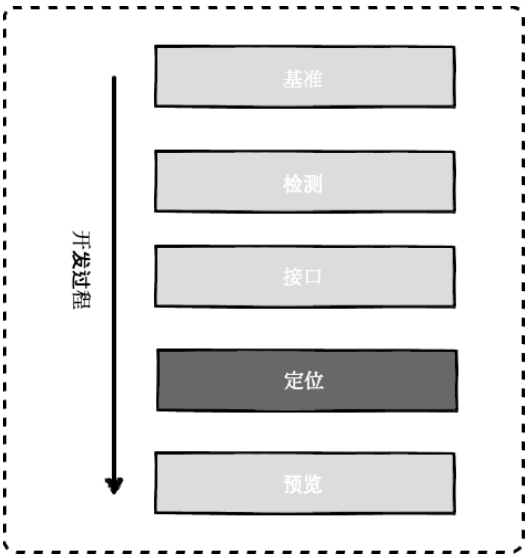
② 仍然不确定 if-parser 需要做到何种程度。

③ IF 中数据 Mock 的需求在可见的时间内不会很强，或者说数据 Mock 这件事情可能不在 IF 的 Scope 中。

④ IF 目前更多关注了业务开发场景下的接口问题，通常有明确的消费者、生产者和监督者。还有一类服务型接口（如各类开放平台）的生产者和监督者容易确定，但是消费者却是不确定的人群，此时由消费者来制定接口定义似乎不易操作，目前更多还是生产者制定接口定义。

相信在持续的工程实践中，以上的问题会不断被解答，答案可能是复杂的，欢迎读者去探索并将你们的成果分享出来。

# 6 定位




1.2 节例子中一个页面在不同终端上 URL 是一致的，而 PC 和 Mobile 页面视图数量上的不匹配会带来本章所述的定位。

来看这个例子。

/page1 是页面地址，用于 PC 和 Mobile 页面，但是 Mobile 的多视图（可以认为每张图片就是一个视图）下如何使用地址直接访问图 6-1 中的第 4 个视图（序号 3）呢？

常用的方案有两种：Hash（或 Hashbang）或 History API。

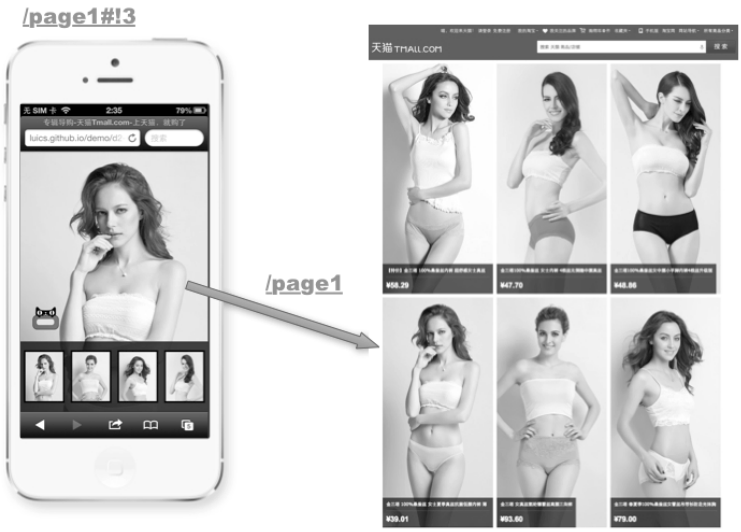



图 6-1　视图映射

# 6.1　定位

## 6.1.1　Hash

下面的 URL 中的字符“#”就是 Hash，字符“#!”是 Hashbang；Hash 或 Hashbang 之后的内容“123”为 hash 值。Hash 字符（或 Hashbang 字符）与 Hash 值共同构成 Hash 串（简称 Hash），如“#!123”：

```
http://www.example.com#!123
```

AJAX 的兴起逐渐导致 OPOA（泛指单页应用）类应用逐渐增多。针对随之而来的单页内的路由问题，在 History API 被广泛支持之前 Hash 成了最合适的方案，由于搜索引擎是不收录 Hash 的，为了解决这个问题 Google 引入了 Hashbang。也就是说使用 Hashbang 既可以实现路由，同时又可以让搜索引擎收录。

用 Hash 进行定位原理很简单，监控“window.hashchanged”事件即可，在部分不支持该事件的浏览器中可以通过轮询 Hash 来兼容。代码 6-1 支持大部分浏览器的

onHashChanged 方法，完整代码可从此处获取[1]。

**代码 6-1 onHashChanged 方法**

```
/**
 * 监听 Hash 变化，支持 IE 6+和其他现代浏览器
 */
function onHashChanged (callback) {
    // IE 8+支持 window.hashchange 事件
    // 这里的判断是简化形式，可改进
    var supportHashChangeEvent = !!window.addEventListener;

    if (supportHashChangeEvent) {
        window.addEventListener ('hashchange', function () {
            callback (location.hash) ;
        }) ;
    }
    else {
        var oldHash = location.hash;
        setInterval (function () {
            var hash = location.hash;
            if (hash !== oldHash) {
                oldHash = hash;
                callback (hash) ;
            }
        }, 100) ;
    }
}

// 监听
onHashChanged (function (hash) {
    alert (hash) ;
}) ;
```

## 6.1.2 History API

HTML5 的 History API（如代码 6-2 所示）允许客户端改写页面 URL，而不触发页面整体刷新，这很好地解决了使用 Hash 导致的 SEO 问题。如图 6-2 所示为 History API 兼容性。

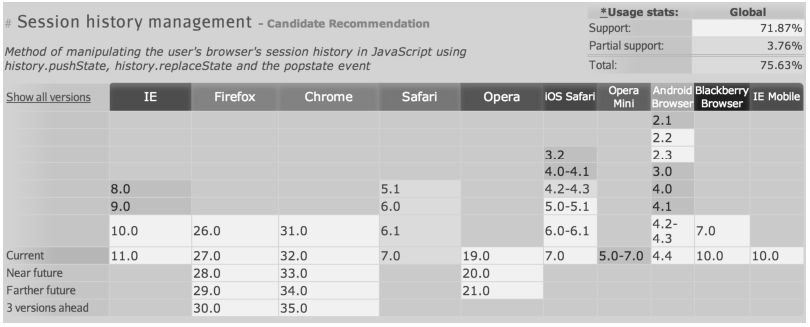
[1] http://luics.github.com/cew/code.html

- history.pushState（state, title, url），为 history 增加一条记录。
  - state，通常设置为{title: title, url: url}。
  - title，记录页面标题。
  - url，记录页面地址。
- window.onpopstate = function（event）{}，在页面回退或前进时触发。
  - event.state，由 pushState 设置的 state 对象。

**代码 6–2　History API**

```
var supportHistoryAPI = !!history.pushState;
if (supportHistoryAPI) {
    var anchors = document.getElementsByTagName('a');
    for (var i = 0; i < anchors.length; ++i) {
        anchors[i].addEventListener('click', function(event) {
            event.preventDefault();
            var title = this.innerHTML;
            var url = this.href;
            console.log(title, url);
            history.pushState({url: url, title: title}, title, url);
        });
    }

    /**
     * 页面首次加载、回退或前进时触发
     */
    window.onpopstate = function(event) {
        var data = event.state;
        console.log('popstate', data, event);
    };
}
```



Session history management - Candidate Recommendation

*Method of manipulating the user's browser's session history in JavaScript using history.pushState, history.replaceState and the popstate event*

| *Usage stats: | Global |
|---|---|
| Support: | 71.87% |
| Partial support: | 3.76% |
| Total: | 75.63% |

| Show all versions | IE | Firefox | Chrome | Safari | Opera | iOS Safari | Opera Mini | Android Browser | Blackberry Browser | IE Mobile |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 2.1 | | |
| | | | | | | | | 2.2 | | |
| | | | | | | 3.2 | | 2.3 | | |
| | | | | | | 4.0-4.1 | | 3.0 | | |
| | 8.0 | | | 5.1 | | 4.2-4.3 | | 4.0 | | |
| | 9.0 | | | 6.0 | | 5.0-5.1 | | 4.1 | | |
| | 10.0 | 26.0 | 31.0 | 6.1 | | 6.0-6.1 | | 4.2-4.3 | 7.0 | |
| Current | 11.0 | 27.0 | 32.0 | 7.0 | 19.0 | 7.0 | 5.0-7.0 | 4.4 | 10.0 | 10.0 |
| Near future | | 28.0 | 33.0 | | 20.0 | | | | | |
| Farther future | | 29.0 | 34.0 | | 21.0 | | | | | |
| 3 versions ahead | | 30.0 | 35.0 | | | | | | | |

图 6-2　History API 兼容性

由于 History API 存在兼容性问题：IE 6～9 和 Android 部分版本不支持。可以通过 History API 和 Hash 构造一个通用的定位功能。

### 6.1.3 视图定位

回到本章开头提到的视图映射，“/page1”是页面 URL。

如果使用 Hash，可以将视图映射为：

```
/page1#!{n}
```

其中“{n}”是自然数（含 0）。

如果使用 History API，可以将视图映射为：

```
/page1-{n}
```

其中“{n}”是自然数（含 0）。

## 6.2 数据

来看另一个和定位相关的问题，图 6-3 显示同一个页面在 PC 上访问时返回整个页面的 HTML，而在 Mobile 上则返回一个 JSON 数据，再由客户端渲染界面。

这需要服务器端有能力根据请求类型（AJAX 或普通 HTTP）返回不同格式的数据。


图 6-3 不同的响应数据格式

### “两次请求”问题

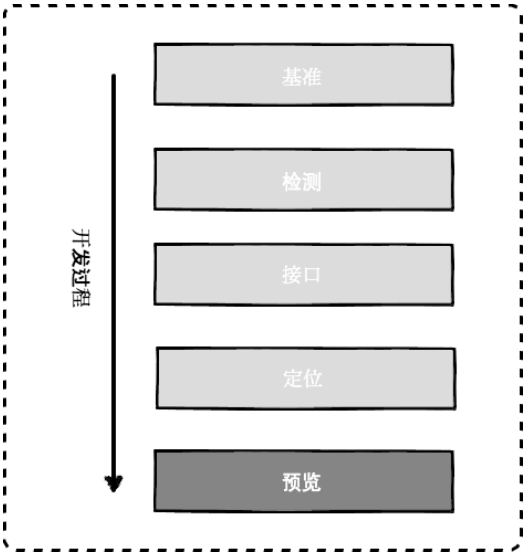
“两次请求”问题是指使用 AJAX 的页面总是需要先加载页面框架后请求数据，再渲染界面，相比一个纯静态页面，AJAX 方式总是多出一次请求。“两次请求”导致了首次展现的延迟，对于大部分页面而言这种延迟是可以接受的，确实存在某些对首次展示延迟非常敏感的场景，如搜索引擎的结果页、电商类的首页和产品展示页。

上面提到的服务器根据请求类型返回合适数据的能力也能用在解决“两次请求”问题中。

1. 首次访问返回 HTML，HTML 由后端模板生成。
2. 非首次访问使用 AJAX 请求 JSON（P）数据，在前端使用同一套模板生成 HTML。这里提到了前后端使用同一套模板，是为了减少同一份逻辑在前后端维护的成本。

# 7

# 预览




1.2 节例子到此已进入到开发的尾声，开发者通常使用多个真机设备预览页面。如果这是一个建站类的工具，用户该如何预览呢？本章介绍了这类预览工具的实现原理，同时给出了核心部分的代码，读者可以扩展它实现自己的跨终端预览。

本书曾提到的 Mobile Emulation 是面向开发者的工具，同时也具有跨终端预览的功能，如图 7-1 所示。参照 Mobile Emulation 的实现机制，构建一个可集成到业务系统中的面向所有用户的跨终端预览工具。

跨终端预览由客户端和服务端两部分构成。

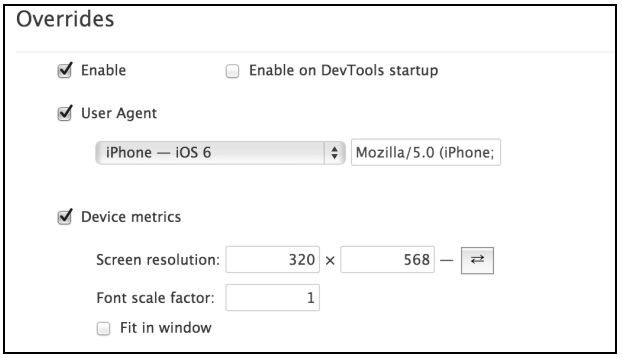



图 7-1　Mobile Emulation（Chrome）

# 7.1　客户端

跨终端预览的客户端功能和 Mobile Emulation 相似，如图 7-2 所示为跨终端预览的客户端界面。客户端包括输入和输出两部分，输入部分包括：

1. UA 设置，可用图形化方式指定一类 UA。
2. 屏幕分辨率设置。
3. 页面地址。

而输出部分通常是尺寸和指定屏幕分辨率相同的 iframe，为了突出预览效果可以添加特定设备的外壳。


图 7-2　跨终端预览的客户端界面

## 7.2 服务端

限于客户端无法改写 UA，需要由服务端执行 UA 重写，并代理 HTTP 的请求与响应。跨终端预览的服务端其实就是一个HTTP Proxy。跨终端预览的基本工作原理如图7-3所示。

**步骤 1**：客户端向服务端发起的请求中携带了请求资源的 URL 和需要改写的 UA。

**步骤 2**：服务端发起资源请求，并改写 UA。

**步骤 3**：服务端获得资源并向客户端返回。

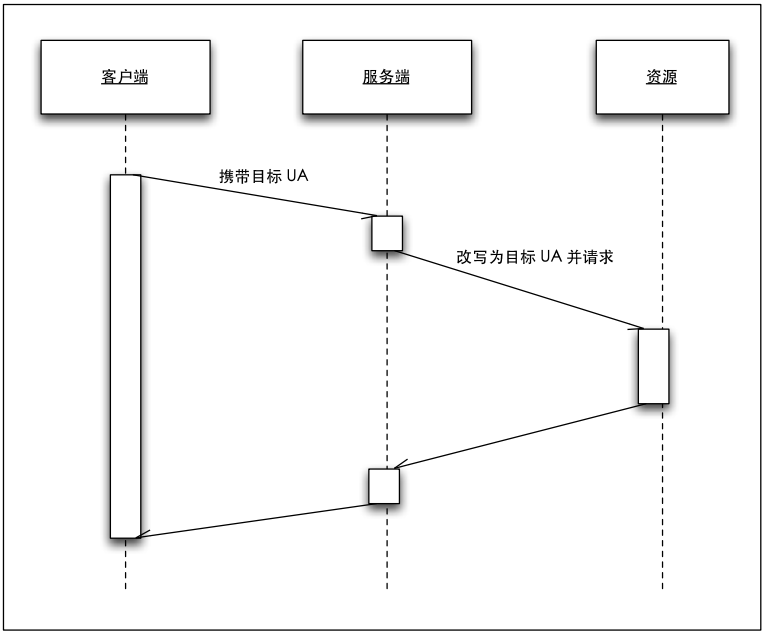



图 7-3 跨终端预览的基本工作原理

代码 7-1 是一个基于 Express 的 HTTP Proxy Server 的简易版本，实现了完整请求、响应过程。

**代码 7–1 HTTP Proxy Server**

```
app.get ('/proxy/:mode', function (req, res, next) {
        var ua = UserAgent[req.params.mode] || req.get ('User-Agent') ;
        var url = 'http://www.tmall.com';
        try {
```

```
            if (req.query.url) {
                url = decodeURI(req.query.url);
            }
        }
        catch (e) {
            url = req.query.url;
        }

        try {
            request.get(url, {
                headers: {
                    'User-Agent': ua
                },
                timeout: 20000,
                encoding: null
            }, function(err, r, body) {
                if (err) {
                    return res.end('无法加载：' + url + ', ERROR:' +
err.toString());
                }
                res.set('Content-Type', r.headers['Content-Type']
|| r.headers['content-type']);
                res.end(body);
            });
        }
        catch (e) {
            res.end('请重试!');
        }
    });
```

完整代码可以从此处[1]获得，如需在本地运行，请按照以下步骤操作。

1. 安装依赖包，在根目录（含 package.json）下运行 “*npm install*”。
2. 启动服务，在根目录下运行“*node app*”。
3. 在浏览器中分别访问，可以看到不同的效果。
   a) http://localhost:3000/proxy/ipad?url=http://www.tmall.com（见图 7-4）
   b) http://localhost:3000/proxy/iphone?url=http://www.tmall.com（见图 7-5）

[1] https://github.com/luics/luics.github.com/tree/master/cew/proxy


图 7-4 使用 iPad UA


图 7-5 使用 iPhone UA

## 7.3　示例

本书不打算详尽介绍跨终端预览页面的设计与实现过程，以下截图展示了跨终端预览的一种实现方式，利用前面两节讲述的客户端与服务端的实现思路可以构建一个属于你自己的跨终端预览工具。如图 7-6 所示为 Phone 的跨终端预览，图 7-7 是 Pad 的跨终端预览，图 7-8 是 PC 的跨终端预览。


图 7-6　Phone 的跨终端预览

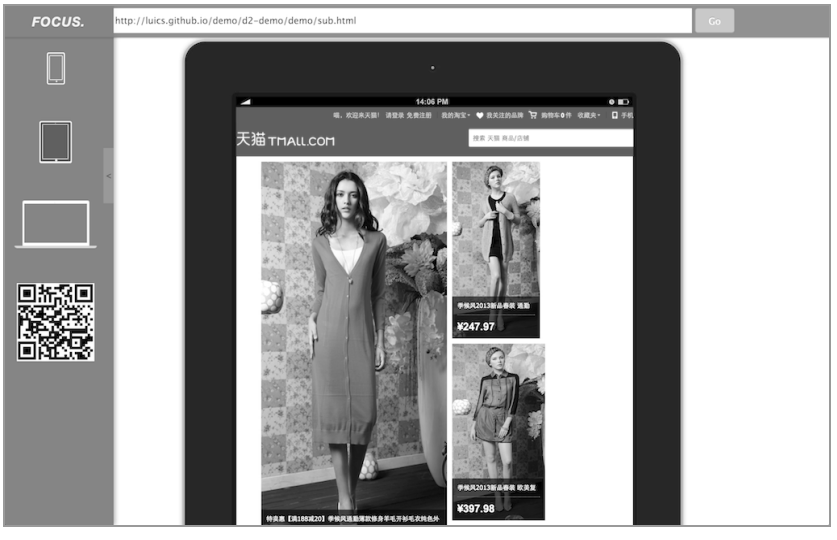



图 7-7　Pad 的跨终端预览


图 7-8 PC 的跨终端预览

# 8 Hybrid App

前面 5 章介绍了开发流程中 5 个重要的技术方案，至此 1.2 节例子也有了雏形。本章讨论的 Hybrid App 已经是一种广为流传的跨终端 Web 的开发方式。

Native App（以下简称 Native）和 Mobile Web（以下简称 Web）二者混合开发的产物被称为 Hybrid App（以下简称 Hybrid）。Hybrid 并不是什么新概念，最早可以追溯到 Symbian 时代，直到 iOS 和 Android 出现之后才充分展现出价值。

## 8.1 Hybrid 简史

### 8.1.1 背景

本书曾提到了“多平台 App”的优势与劣势。Hybrid 既利用了 Native App 丰富的设备 API（Device API），又能拥有 Mobile Web 的跨平台、高效开发、快速发布的能力，对于相当庞大的应用场景而言都是适用的。

Hybrid 优势在于：

**跨平台**

Web 内容可以做到开发一次，所有平台生效，诸多产品需要这种能力。

**快速发布**

iOS 平台，Apple Store 平均审核周期 1～2 周不等，甚至更长，产品的发布周期从 2 周到 1 月，这对需要快速发布的产品而言难以接受。

Android 平台，应用商店众多，发布过程烦琐。虽然可以应用内升级，但是带来的问

题是新 App 需要通过应用商店，此外 APK 体积庞大，2G/3G 环境下体验差。

**高效开发**

Web 开发经过 20 年的发展，已经将结构（HTML）、表现（CSS）、行为（JavaScript）3 部分很好地分离开，在分工协作、开发效率上会具明显优势。

**丰富的 Device API**

Web（HTML5）强调通用性，受限于标准和浏览器实现，许多有用的系统功能未能得到支持（或部分支持）。而 Native 最大的优势在于设备 API 的调用能力，只要桥接 Native 和 Web，Web 也就能够拥有这种能力。

Hybrid 劣势表现为：

1. CPU/GPU 密集类应用目前看更适合 Native，例如极品飞车这样的游戏。这种劣势是在不断弱化的，正如 “CSS Transform 3D”引入 GPU 大大缓解了 Web 动画不流畅的问题。
2. 静态资源从服务器端加载导致的 UI 展示延迟问题。这个问题可以通过 Native 拦截 WebView 通信加载已打包的公共库来缓解。

## 8.1.2　简史

**雏形**

雏形阶段大致为：

- Symbian V3/5 时代已经有 Hybrid 雏形。
- iOS 最初的 App 都是由 Objective-C 编写而成的，受限应用商店的发布周期，内容经常变化的部分开始通过使用内置浏览器控件（WebView）加载服务端页面来实现。
- Android 出现并流行之后，可以将更多的 App 功能通过 Hybrid 来实现，这样在不同平台上就可以只维护一个版本。

**发展**

“跨平台”成了 Hybrid 最大的卖点，以 PhoneGap[1] 为首的 Hybrid 框架陆续出现，

[1] PhoneGap 是主流 Hybrid 框架。

带来了诸多改变。

- 访问设备功能。
  - Web（HTML5）不支持的功能可以让 Native 实现，再通过 Native 和 Web 之间通信，通过这种方式可以让 Web 获得和 Native 相同的设备 API 调用能力，这是 PhoneGap 这类 Hybrid 框架的基本工作原理。
  - 与此同时，将 Web 代码转为 Native 的 Hybrid 框架（如 Tianium[2]）也出现了。
- PhoneGap 子项目 weinre 是一种远程调试工具，极大地缓解了 Hybrid 难于调试的问题，进一步促进了 Hybrid 的发展。
- Hybrid 框架提供了应用打包功能，开发者可以完全使用 HTML、CSS、JavaScript 开发 Native App。

**成熟**

随着 PhoneGap 这类 Hybrid 框架在全球的流行，一些问题暴露了出来，也正是这些问题的解决，让 Hybrid 走向成熟。

- 开发体验提升。
  - weinre 这类调试工具仍属于插件性质，诸如“网络”、“本地资源”等高级调试功能无法支持，WebView 的原生调试需求越来越强烈。
  - iOS 6.0+已经支持原生的远程调试[3]。
  - Chrome for Android 在原生远程调试上处于领先地位[4]。
  - 从 Android 4.4 开始，WebView 也支持原生的远程调试[5]。
- 提升 WebView 性能的呼声日益增强。
- 某些追求极致性能的功能转由 Native 实现，如转场（页面间切换）动画。
- 静态资源本地化是理想状态，其他场景下 Native 拦截 WebView 的请求，并让公共资源重定向到 App 内置资源，同样能实现为 Web 提速。

---

2 http://www.appcelerator.com/titanium/

3 http://www.36kr.com/p/117773.html

4 http://www.html5rocks.com/en/tutorials/developertools/mobile/

5 https://developers.google.com/chrome-developer-tools/docs/remote-debugging#debugging-webviews

### 8.1.3　现状

以上便是 Hybrid 的发展概述，从国内最新的资料可以看出，Hybrid 的趋势也是非常明显的。从图 8-1 可以看到越来越多的开发者决定使用 Hybrid（跨平台技术），最近两年的总量已经有 54%；而接近 60%的开发者在 Hybrid 的技术方案上选择了 PhoneGap。

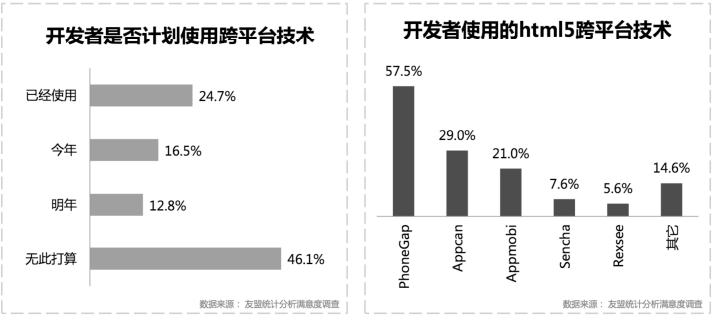



图 8-1　Hybrid 在国内的发展情况[6]

KendoUI 2013 年面向开发者做了一次有关的 Hybrid 调查，并将结果整理成了报告 *The HTML5 vs. Native Debate is Over and the Winner is...*[7]。下面来看报告中几个关键的结论。

1. 在受访的 2309 个 Mobile 开发者中，到 2013 年 8 月为止完全使用 Native 开发的只有 8%，而剩余的 92%都可以被认为使用的是 Hybrid，如图 8-2 所示。

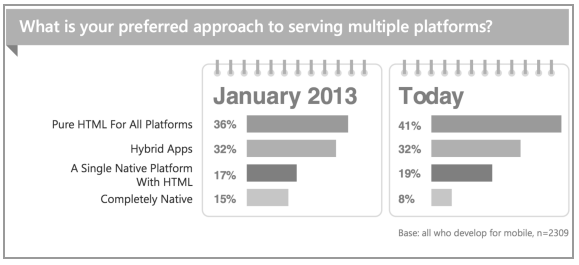



图 8-2　Hybrid 使用情况

[6] 摘自《友盟 2013 上半年报告》2013.09。

[7] http://www.kendoui.com/surveys/html5-native-debate-is-over.aspx

2. App 的跨平台特性成为一个重要的考虑，如图 8-3 所示。

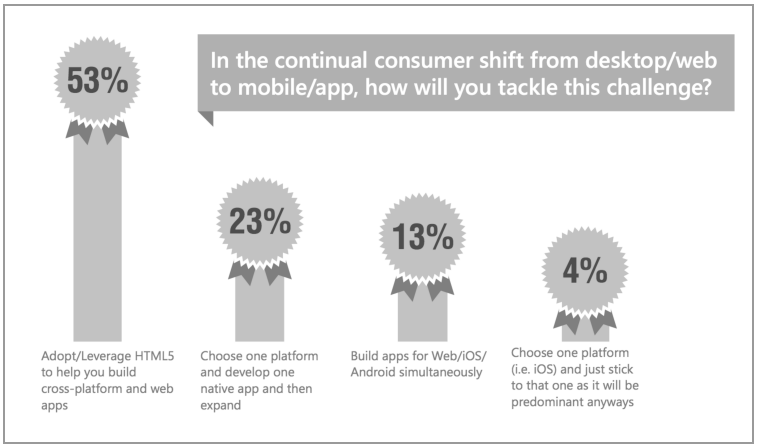



图 8-3 跨平台特性受关注

图 8-4 显示了 Hybrid 惊人的增长速度：2013 年无论是开发中、已发布的 Hybrid（或 HTML App）均相比于 2012 年出现了超过 125%～400%的增长率[8]。

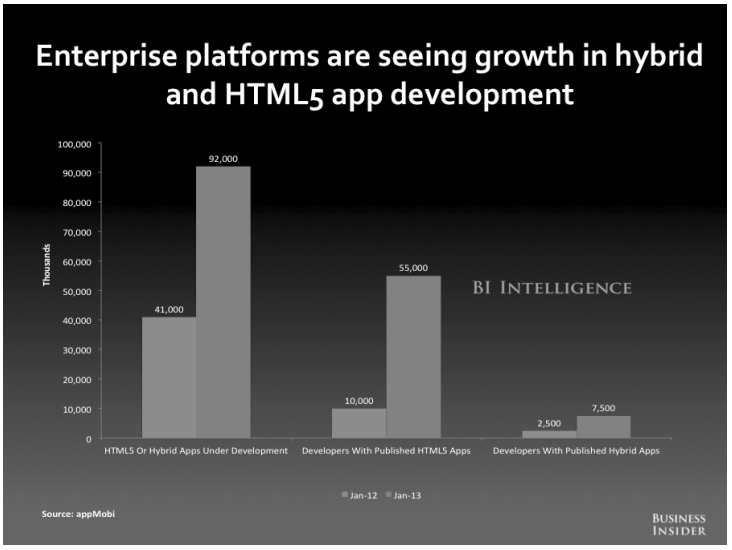



图 8-4 Hybrid 增长迅猛[9]

[8] http://www.businessinsider.com/html5-vs-native-apps-for-mobile-2013-6?op=1

[9] 此图来自 http://www.businessinsider.com/html5-vs-native-apps-for-mobile-2013-6?op=1。

## 8.2　Hybrid 技术

无论 Android 还是 iOS，实现一个最简单的 Hybrid App 只需要几行代码：实例化 WebView、加载页面，之后便是页面自身的代码。要想实现更为复杂的、完整的 Hybrid 还需要不少知识。

1. Mobile Web 开发基础：HTML、CSS、JavaScript。
2. Native App 开发基础：Android、iOS。
3. Native 与 Web 双向通信机制。

Mobile Web 开发基础可以参考本书第 2 章，Native App 开发基础已经超出本书的讨论范围，同样有很多可选择的书籍，本节来讲剩余的第 3 个问题 “Native 与 Web 双向通信机制”。

### 8.2.1　Native 调用 Web

无论 Android 还是 iOS ，Native 调用 Web（JavaScript） 都有很好的原生支持，如代码 8-1 和代码 8-2 所示。Android 中的调用方式如下，其中 webView 是 Webview 的实例。

**代码 8–1　Android 调用 JavaScript**

```
webView.loadUrl("javascript:(function(){alert('ok');})()");
```

iOS 中的调用方式如下，其中 webView 是 UIWebview 的实例。

**代码 8–2　iOS 调用 JavaScript**

```
[webView stringByEvaluatingJavaScriptFromString: @"alert('ok')
" ];
```

### 8.2.2　Web 调用 Native

“Native 调用 Web”本质上是 JavaScript 脚本的动态执行，在“Web 调用 Native”的场景下由于目前 Native 语言（Java 和 Objective-C）不容易像 JavaScript 那样便于动态执行，所以需要另辟蹊径。

#### 8.2.2.1　Android

Android 上常见的方式有 3 种。

1. 重写 WebViewClient.shouldOverrideUrlLoading（如代码 8-3 所示）。

**代码 8–3 重写 WebViewClient.shouldOverrideUrlLoading**

```
webView.setWebViewClient (new WebViewClient () {
      @Override
      public boolean shouldOverrideUrlLoading (WebView view, String
url) {
          // TODO 解析 URL 并触发 Native 代码
return true;
      }
  }) ;
```

当页面内的URL发生变化时，如点击链接、执行JavaScript（如 *location.href="http://"*）等均会触发 WebViewClient.shouldOverrideUrlLoading，通过将 Web 调用 Native 的数据封装在 URL，再由 Native 解析数据并执行响应 Native 方法。

2. 重写 WebChromeClient.onJsPrompt，或 onJsConfirm，或 onJsAlert，以 WebChromeClient.onJsPrompt 为例，如代码 8-4 所示。

**代码 8–4 重写 WebChromeClient.onJsPrompt**

```
 webView.setWebChromeClient (new WebChromeClient () {
      public boolean onJsPrompt (WebView view, String url, String
message, String defaultValue, JsPromptResult result) {
      // TODO 解析 message 并触发 Native 代码
          result.confirm ("") ;
          return true;
      }
  }) ;
```

当执行“window.prompt（“{}”）”这样的 JavaScript 代码时，将会触发 WebChromeClient.onJsPrompt。

3. WebView.addJavascriptInterface，这种方式和前两种都不同，通过将 Java Object（A） 映射为 JavaScript Object（B），从而调用 B.func1 时将会自动触发 A.func1，通过这种原生的方式实现了 “Web 调用 Native”，如代码 8-5 所示。

**代码 8–5 WebView.addJavascriptInterface**

```
 webView.addJavascriptInterface (new Object () {
      public void func1 () {
      }
```

```
        public void func2 () {
        }
    }, "webViewObj") ;
```

以上 3 种方式，最常用的是方式 2；方式 2 相比方式 1 有内置的队列支持，不会出现高频访问数据丢失的情况；方式 3 是 Android 原生方式，但是不如前两种方式灵活。

#### 8.2.2.2　iOS

iOS 中可用的方式类似 Android 中的 WebViewClient.shouldOverrideUrlLoading， 通过监控 WebView 的 URL 变化实现 Web 调用 Native，如代码 8-6 所示。

**代码 8–6　shouldStartLoadWithRequest**

```
- (BOOL)webView:(UIWebView *) webView shouldStartLoadWithRequest:
 (NSURLRequest *) request
navigationType: (UIWebViewNavigationType) navigationType {

}
```

### 8.2.3　Bridge

有了前两节的知识，可以实现一个通用模块（Bridge）来维护不同平台上的“Web 与 Native 双向通信机制”功能。如图 8-5 所示为 Web 调用 Native 的 Bridge 时序图。

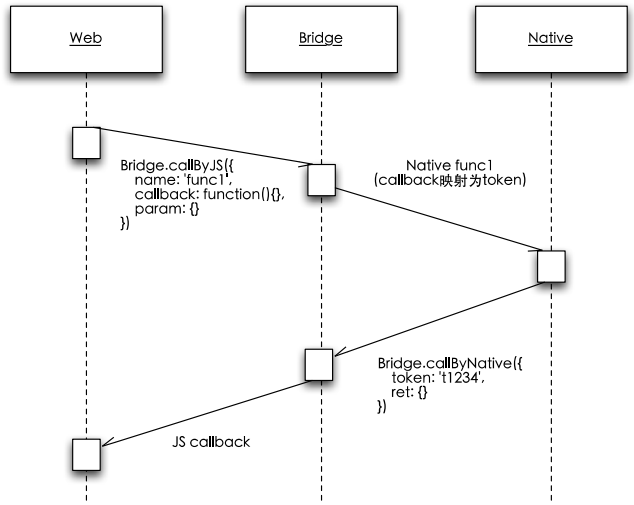



图 8-5　Web 调用 Native 的 Bridge 时序图

Web 调用 Native 的实现原理如下。

1. Web 端调用 Bridge.callByJS（{name:'func1', callback: function（）{}, param:{}}），由 Bridge 根据特定“Web 调用 Native”方式通知 Native 执行相应方法（图 8-5 中的“func1”）。
2. Native 执行完毕后通过“Native 调用 Web”的方式调用 Bridge. callByNative（{token: 't1234' }）。如图 8-6 所示为 Native 调用 Web 时的 Bridge 时序图。
3. 其中 JavaScript 回调函数会映射为字符串型的 token，通过这个方式来保证最终触发 JavaScript 的回调函数（包括匿名函数和通过闭包实现的私有函数）。

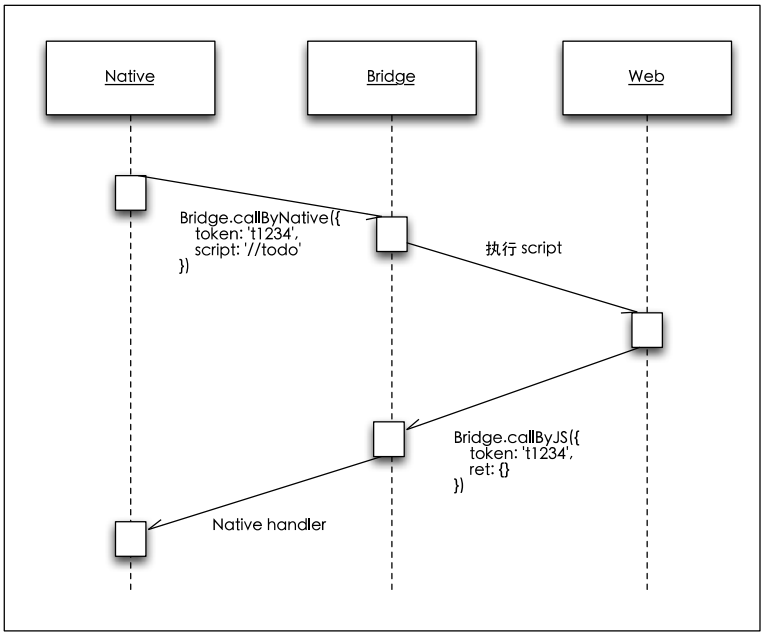



图 8-6 Native 调用 Web 时的 Bridge 时序图

可以看到，Bridge 实现“Native 调用 Web”是类似的。

1. Native 端调用 Bridge.callByNative（{token:'t1234', script: '//todo'}），由 Bridge 根据特定“Native 调用 Web”方式通知 Web 执行相应脚本。
2. Web 执行完毕后通过“Web 调用 Native”的方式调用 Bridge. callByJS（{token: 't1234' }）。

3. 如果 Bridge.callByNative 的 script 中执行了异步操作，需要在 script 主动调用 Bridge.callByJS，并且不需要传 token 参数。

笔者已经在 Android 上实现了完整的 Bridge[10]，Bridge 由 JavaScript 实现可以运行在 Android 和 iOS 的 WebView 中，同时也非常容易扩展到 Windows Phone 等新平台，如代码 8-7 所示。

1. Bridge 代码在产品环境下使用时请设置 DEBUG = false。
2. 避免在 iOS 下快速变化 URL 时造成的数据丢失，可以考虑使用队列机制缓存命令。
3. 扩展至 Windows Phone 等平台时 JavaScript 部分只需要扩展 invoke，Native 代码可以参考 Android 的实现。
4. 目前 Bridge 单次通信后会删除回调函数，如果需要多次调用缓存的回调函数（如连续监控传感器数据），可以扩展 Bridge.callByNative。

**代码 8–7　bridge.js**

```
(function (window) {
   var DEBUG = true;
   var callbacks = {};
   var guid = 0;
   var ua = navigator.userAgent;
   // TODO 精确性待改进
   var ANDROID = /android/i.test (ua) ;
   var IOS = /iphone|ipad/i.test (ua) ;
   var WP = /windows phone/i.test (ua) ;
   //ANDROID = 0; IOS = 1;

   /**
    * 方便在各个平台中看到完整的 log
    */
   function log () {
      if (DEBUG) {
         console.log.call ( console, Array.prototype.join.call
(arguments, ' ') ) ;
      }
   }
```

[10] http://luics.github.io/cew/Bridge.zip

```
/**
 * 平台相关的 Web 与 Native 单向通信方法
 */
function invoke (cmd) {
    log ('invoke', cmd) ;
    if (ANDROID) {
        prompt (cmd) ;
    }
    else if (IOS) {
        location.href = 'bridge://' + cmd;
    }
    else if (WP) {
        // TODO ...
    }
}

var Bridge = {
    callByJS: function (opt) {
        log ('callByJS', JSON.stringify (opt) ) ;
        var input = {};
        input.name = opt.name;
        input.token = ++guid;
        input.param = opt.param || {};
        callbacks[input.token] = opt.callback;

        invoke (JSON.stringify (input) ) ;
    },
    callByNative: function (opt) {
        log ('callByNative', JSON.stringify (opt) ) ;
        var callback = callbacks[opt.token];
        var ret = opt.ret || {};
        var script = opt.script || '';

        // Native 主动调用 Web
        if (script) {
            log ('callByNative script', script) ;
            try {
                invoke (JSON.stringify ({
                    token: opt.token,
                    ret: eval (script)
                }) ) ;
            } catch (e) {
                console.error (e) ;
            }
        }
        // Web 主动调用 Native，Native 被动响应
```

```
            else if (callback) {
                callback(ret);
                try {
                    delete callback;
                    log(callbacks);
                } catch (e) {
                    console.error(e);
                }
            }

        }
    };

    window.Bridge = Bridge;
    window.__log = log;
})(window);
```

## 8.3　Hybrid 框架

目前一个 Hybrid 框架通常提供以下功能。

1. Device API：封装 Native 的功能，跨平台提供一致的 Device API。
2. App 打包：将 HTML5 编写的代码打包为 App（Titanium 会转换代码）。

PhoneGap 几乎成了 Hybrid 的代名词，Titanium 和 PhoneGap 的设计理念差异较大，图 8-7 形象地展示了 PhoneGap 和 Titanium 的组成部分。

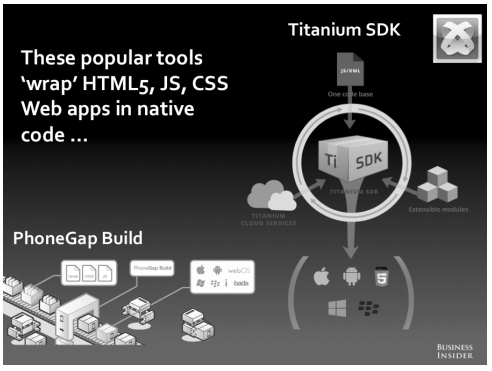



图 8-7　Hybrid 框架[11]

[11] 此图来自 http://www.businessinsider.com/html5-vs-native-apps-for-mobile-2013-6?op=1。

## 8.3.1 PhoneGap

### 8.3.1.1 PhoneGap 和 Cordova

PhoneGap 开发商 Notibi 2010 年将 PhoneGap 代码贡献给 Apache 软件基金（ASF），PhoneGap 核心引擎成为新的开源项目 Cordova，同时 PhoneGap 成了 Cordova 的一个发行版本[12]。2011 年 10 月，Notibi 被 Adobe 收购[13]，但没有影响到 PhoneGap 和 Cordova 的开源性质。

### 8.3.1.2 原理

```
written once, run everywhere
```

如引文所述“一处编写，多处运行”，PhoneGap 主要的功能为：

1. 提供 Hybrid API，可由 JavaScript 直接调用诸如加速度、摄像头、指南针、GPS、联系人等系统级 API，完整的 API 列表请访问 PhoneGap API Reference。
2. 使用 Web（HTML、CSS、JavaScript）开发的内容经过 PhoneGap 编译打包为各个平台的 Native App，如图 8-8 所示。


图 8-8 PhoneGap 编译打包功能

### 8.3.1.3 经典案例

来自 PhoneGap Showcase[14]和其他数据源的资料显示：

- Facebook Mobile SDK[15]和 SalesForce Mobile SDK[16]均是基于 Cordova 的分支

[12] http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/

[13] http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html

[14] http://phonegap.com/app/

[15] https://developers.facebook.com/docs/guides/mobile/

[16] http://wiki.developerforce.com/page/Mobile_SDK

开发的。

- Facebook 客户端中 Web 代码超过 90%[17]。
- LinkedIn iPad 客户端中 Web 代码甚至超过 95%。
- Wikipedia 更是直接用 PhoneGap 开发了自己的 iOS/Android Hybrid App[18]，并将代码在 GitHub 上开源 [19]。

## 8.3.2 Titanium

Titanium 设计思路和 PhoneGap 有很大不同，Titanium 目的为移动开发提供一种跨平台的 JavaScript 运行时环境和 API。

### 8.3.2.1 设计思路

Titanium 设计的核心思路如下。

1. 有一套核心的移动开发 API，它们可以跨平台进行规范，这些方面的重点应放在代码重用上。
2. 有针对特定平台的 API、用户界面约定以及功能特性，开发者在针对该特定平台从事开发时采用，应该有针对特定平台的代码，以便这些用例提供最佳的用户体验。

Titanium 从设计理念上不追求“written once, run everywhere”，这是它的缺点，但同时它追求平台差异的更佳的用户体验，因而也受到一部分用户的追捧。Titanium 的另一个缺陷是插件难于扩展，要想支持新平台则更加困难。

### 8.3.2.2 工作流程

工作流程如下。Titanium 工作流如图 8-9 所示。

1. 使用 Titanium SDK 在自带的 IDE（ALLOY）中开发。

---

[17] http://www.geekpark.net/read/view/164456

[18] http://itunes.apple.com/us/app/wikipedia-mobile/id324715238?mt=8

[19] https://github.com/wikimedia/WikipediaMobile

2. 使用工具编译为平台相关的 App。

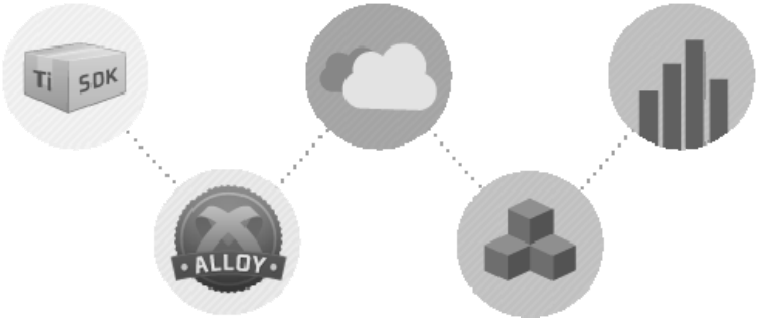



图 8-9 Titanium 工作流

## 8.4 Device API

无论 PhoneGap 还是 Titanium，Device API 都是其核心要素之一。结合 PhoneGap/Cordova API[20]、Titanium SDK[21]、W3C Device API[22]相关的数个工作组的成果，以及 Android[23]和 iOS[24]设备传感器的特性，整理出下面的表格。

表 8-1 的一些说明如下。

1. 数据更新于 2014-01-20。
2. ↗——表示暂无支持。
3. 国内 Geolocation API 在 Android 内置浏览器上获取地理位置反编码信息经常失败。
4. 加速度计、陀螺仪、磁力计，三者都是硬件传感器，通常不直接使用他们获取的原始参数，而是通过相互组合计算后得到更有意义的参数。
   - 以导航为例，至少需要使用：GPS、磁力计，GPS 负责经纬度定位、

[20] http://docs.phonegap.com/en/3.3.0/#API%20Reference

[21] http://docs.appcelerator.com/titanium/3.0/#!/api

[22] W3C Device API Working Group: http://www.w3.org/2009/dap/
W3C Geolocation Working Group: http://www.w3.org/2008/geolocation/

[23] Android Sensors Overview: http://developer.android.com/guide/topics/sensors/sensors_overview.html

[24] Basic Sensors in iOS: http://shop.oreilly.com/product/0636920021162.do

磁力计提供方向引导、陀螺仪（可选）可以在导航中提供 3D 坐标系支持。

5. 扬声器和视频播放（功能）不是传感器，放在表中是和传声器、摄像头对比。
6. 近场通信在日、韩、欧美已经应用较广，国内还处在基础设施建设阶段[25]。

表 8–1　Device API

| 设备功能 | 标识 | 核心参数 | 典型应用场景 | Web 标准 | Hybrid 支持案例 |
|---|---|---|---|---|---|
| 硬件特性 | | | | | |
| GPS | Geolocation | 经纬度 | 搜索<br>地图/导航<br>社交<br>电商 | W3C Geolocation API | PhoneGap Geolocation |
| 加速度计 | Accelerometer | *xyz* 轴加速度 | “摇一摇”（shake）<br>导航<br>游戏 | W3C Motion API | PhoneGap Accelerometer<br>PhoneGap Compass |
| 陀螺仪[26] | Gyroscope | *xyz* 轴角速度及偏移角 | 指南针<br>地图/导航<br>赛车类游戏<br>增强现实 | W3C Motion API | 〃 |
| 磁力计 | Magnetometer | *xyz* 轴磁场强度 | 指南针<br>导航 | W3C Motion API | PhoneGap Compass |

[25] http://www.zhihu.com/question/20277199/answer/14586392

[26] http://www.dianshang365.cn/sjzs/154.html

续表

| 设备功能 | 标识 | 核心参数 | 典型应用场景 | Web 标准 | Hybrid 支持案例 |
|---|---|---|---|---|---|
| 距离传感器 | Proximity | 距离 | 通话过程锁屏 | W3C Proximity Events | ↑ |
| 传声器 | Microphone | 音频数据 | “咻一咻”<br>语音录制<br>语音识别（输入） | W3C Audio API | PhoneGap Capture |
| 扬声器 | Speaker | | 音频播放 | W3C Audio API | - |
| 摄像头 | Camera | 视频数据 | 拍照<br>视频录制<br>二维码<br>图像识别（增强现实） | W3C Video API | PhoneGap Camera<br>PhoneGap Capture |
| 视频播放 | Video | | 视频播放 | W3C Video API | - |
| 光线传感器 | Light | 照度 | 游戏<br>环境适配 | W3C Light Events<br>W3C MQ Luminosity | ↑ |
| 气压传感器 | Pressure | 气压 | 监控气压<br>测量海拔 | W3C Pressure Events | ↑ |
| 气温传感器 | Temperature | 气温 | 监控气温 | W3C Temperature Events | ↑ |
| 湿度传感器 | Humidity | 湿度 | 监控相对/绝对湿度 | W3C Humidity Events | ↑ |

续表

| 设备功能 | 标识 | 核心参数 | 典型应用场景 | Web 标准 | Hybrid 支持案例 |
|---|---|---|---|---|---|
| 振动器 | Vibrator | | 提醒功能 | W3C Vibration API<br>支持列表 | ↑ |
| 近场通信 | NFC | | 支付<br>门禁 | | ↑ |
| 软件特性 | | | | | |
| 设备事件 | Events | | page visibility<br>battery<br>online/offline | W3C Network Information API<br>W3C Page Visibility API | PhoneGap Events |
| 网络类型 | Connection | 网络类型 | 大数据下载提醒 | W3C Network Information API | PhoneGap Connection |
| 通信录 | Contacts | | | | PhoneGap Contacts |
| 文件 | File | | 访问文件系统 | W3C File API | PhoneGap File |
| 应用内浏览 | InAppBrowser | | | | PhoneGap InAppBrowser |
| 通知 | Notification | | 通知 | W3C Web Notifications | PhoneGap Notification |
| 设备标识 | Device | | | | PhoneGap Device |

## 8.4.1　动作传感器

动作传感器包括陀螺仪、加速计、距离传感器。

#### 8.4.1.1 陀螺仪

陀螺在高速旋转时有非常好的稳定性，这种特性可用于对偏转角度的测量，如图 8-10 所示。最初的实体陀螺仪就是应用了这个原理制造出来的，此后的电子陀螺仪沿用相似的原理并植入到导航设备中。今天的大部分移动设备中也沿用该原理。

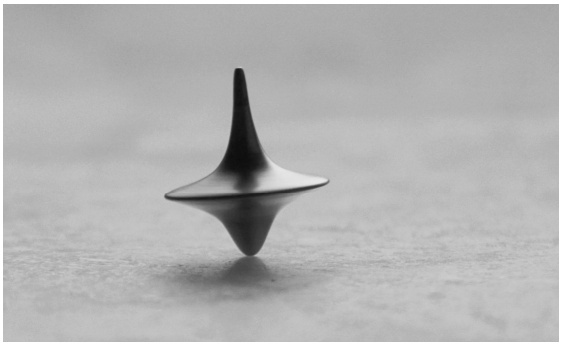


图 8-10 旋转中的陀螺[27]

图 8-11 展示的“春分竖蛋游戏”需要用户调整手机以保持蛋能够立在小猫尾巴上，用到了陀螺仪。


图 8-11 春分竖蛋游戏

[27] 来自电影《盗梦空间》。

另一个陀螺仪的典型案例是：iOS 7（iPhone 或 iPad）的桌面壁纸会随着用户轻微的翻转而出现位移，带来一种很特别的体验。

#### 8.4.1.2　加速计

加速计获取的原始参数包含重力加速度，这也是加速计被称为“重力感应”的原因；应用中通常需要过滤掉重力加速度，分离后的数据通常被称为线性加速度。人们往往容易混淆加速计和陀螺仪，因为二者都检测运动过程中的某些参数。

早先加速计被安装在一些游戏手柄中，并在球类游戏中大放异彩。如今移动设备中加速计也成了标配，就像图 8-12 显示的那样，已经可以做到用手机替代专用的游戏手柄参与球类游戏。

加速计最广为人知的案例便是“摇一摇”，出现在注入社交、即时通信、电商活动、移动支付等各类应用中。

图 8-12　加速计应用于球类游戏

#### 8.4.1.3　距离传感器

距离传感器在功能机时代已有，用于监控与外部物体的距离；通常用于通话过程中监控人脸是否贴近话机，如贴近则锁屏防止误操作。

### 8.4.2　环境传感器

环境传感器包括 GPS、磁力计、光线传感器、气压传感器、气温传感器、湿度传感器等。

- GPS 是全球定位系统（Global Positioning System）的简称，用于提供实时、

全天候和全球性的导航服务，最典型的应用便是地图、导航。

- 磁力计可获取磁场强度，典型的应用是指南针、汽车导航，如图 8-13 所示。
- 光线传感器，可用于感知环境光线强度的变化，从而改变当前应用的某些行为；最典型的应用是大部分移动设备会根据光强度改变屏幕亮度，以保持最舒适的阅读效果；同样还可以根据环境光强改变应用的皮肤，Media Queries Level 4[28]中引入了 light-level，不久的将来通过 CSS 也能够实现类似的功能。
- 气压/气温/湿度传感器分别获取空气的压力、温度和湿度信息，目前的应用和关注较少。


图 8-13 磁力计在指南针和导航中应用

## 8.4.3 音频

通过传声器获取实时的语音输入，能做很多有趣的事情。

- 获取声音强度，作为游戏的输入数据。

[28] http://dev.w3.org/csswg/mediaqueries4/#light-level

- 加密录制的音频数据用于移动支付等应用场景，如图 8-14 所示。
- 将录制的语音通过服务器识别后实现的语音输入或其他控制。


图 8-14　声波支付

### 8.4.4　视频

视频是更为庞大和复杂的传感器应用，从最简单的拍照、视频录制，到更为复杂神奇的 AR（增强现实），随着 AR 技术的成熟，越来越多的 AR 被引入到移动应用中。AR 与 VR（虚拟现实）是密不可分的，就在笔者写下这段文字时，一个由游戏巨头和硬件厂商发起的“沉浸式技术联盟”[29]的 VR 组织成立了，VR 界的领军人物 Carmark 认为两年内 VR 技术会得到大范围普及。

## 8.5　小结

回顾 PC 时代的 20 多年，Web 的发展异常迅猛，但桌面应用开发并未消失，仍有其适用的领域。今天的 Mobile 开发才刚起步，Native App 或许只是一种技术上的过渡形态，但在未来相当长的一段时间内，各平台应用商店仍然是通往最终用户的重要渠道，App 也会一直存在下去。在这样的背景下，Hybrid 成了一种很特别的存在形态，它既是 Native App 又具有 Web 的基因，关于 Hybrid 的探讨和研究会一直进行下去，想必也会发生些更有趣的事情。

---

[29] Immersive Technology Alliance http://www.engadget.com/2014/03/13/immersive-technology-alliance/

# 9 存储

本章相对前面几章较独立，介绍了一种跨域的存储方案，并且这种方案能够适用于跨终端的场景。

2013 年 10 月阿里集团内部进行了一次 Kissy Gallery 组件大赛，参赛的组件达到 75 个，如图 9-1 所示为本次组件大赛结果。本章讲述的存储组件（Storage）拿到了第 3 周的“本周之星”[1]和总决赛的冠军。本章并未列在 2013 年 8 月最初的大纲中，由于 Storage 本身与跨终端 Web 的关联性以及 Storage 的受认可程度，笔者还是将其单列一章。

| | frameCartoon<br>妙净 | imageFilter<br>加里 | storage<br>鬼道 | Mock<br>墨智 | SnakeSlider<br>阿古 | musicPlayer<br>韩升 | albums<br>翰文 |
|---|---|---|---|---|---|---|---|
| 评委评分 | 57.28 | 58.85 | 64.71 | 66.14 | 62.14 | 63.71 | 61.71 |
| 社区评分 | 11.76 | 25 | 22.55 | 4.41 | 6.37 | 6.86 | 12 1/4 |
| 总分 | 69.04 | 83.85 | 87.26 | 70.55 | 68.51 | 70.57 | 73.96 |

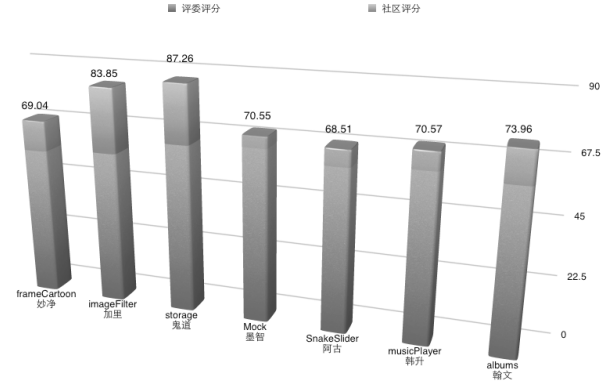



图 9-1　组件大赛结果

[1] 笔者发表的一篇短文《使用 Storage 的 3 个理由》http://blog.jobbole.com/49881/。

Storage[2]是一个跨终端、跨域的存储组件。

1. 十亿级日调用次数的考验，稳定可靠。
2. 兼容 IE 6+、Chrome、Firefox、Safari。
3. 不使用 Flash 方案，完美支持移动浏览器。
4. 跨子域、主域的数据存取，且不改写 document.domain。
5. 支持 Object、Array 等复杂对象存取。

## 9.1　状态持久化

随着页面复杂性急剧增加，以及页面间关联性不断提高，原来看似无关联的多个页面间某些状态同步的需求在不断增多[3]。

状态数据存放于后端带来的直接问题是，海量的状态信息给服务器（无论是存储容量还是网络 I/O）带来了更大的挑战；前端存储状态的天然优势在于利用亿级客户端设备存储状态信息，几乎无网络开销（可能会有同步）。简单计算下，10 亿个客户端按照每个 2MB 来算就是约 2000T。以笔者最近接触到的某项目为例，状态存储在前端直接节约了 500 台缓存服务器；另一个例子是，百度、Google 的搜索历史保存在前端，用户输入时可以快速地从本地存储中获得搜索建议。

使用 Cookie 存储标记状态的行为由来已久，可以看作是前端保存状态最简单的形态。受限 Cookie 的长度（4KB）以及 Cookie 对网络请求造成的额外负担，通常我们只用 Cookie 保存那些开关量；Storage 使用 localStorage（IE 6/7 使用 userData）存储数据，存储容量远超 Cookie，也不会加重网络开销。

Cookie 实际能存储的数据很少（4KB），并且考虑到 Cookie 数据存在于 HTTP 请求和响应 Header 中会加重网络开销；flash storage 和 userData 都属于“外挂”资源，对于前端而言是个黑箱，随着移动端的兴起，这些技术也将退出历史舞台；localStorage 是 HTML5 标准的一部分，无论在存储上限还是对网络的影响上都是更

---

2　http://gallery.kissyui.com/storage/1.1/guide/index.html

3　本节最初讨论见于笔者博客：https://github.com/luics/luics.github.com/issues/4。

佳的选择，目前几乎支持所有的浏览器（IE 6/7 除外）。

可是出于安全考虑，localStorage 存储的数据不能跨子域访问（即使修改 document.domain）；Storage 将数据存储在代理页（b.com）所在域，宿主页（a.com）通过与代理页的跨域通信存储数据，跨域通信使用的 postMessage（IE 6/7 使用 window.name）不会改写 document.domain。前端可能会用 document.domain 进行环境判断，而改写 document.domain 在复杂系统中很容易产生隐患，这种隐患都是开关级别的，通常比较严重；即使没有这种隐患，document.domain 改写也只能在同一个主域下进行，对于不同主域（如 a.com 和 b.com）的情况就无能为力了。

iOS 平台不支持 Flash，随着 Adobe 宣告不再支持 Flash 移动端[4]，localStorage（及 DB 类存储）实际成了移动端上唯一可用的存储介质。 Storage 不使用 Flash 方案，一份代码可运行在 PC 端和移动端的浏览器上。

# 9.2 技术方案

## 9.2.1 整体方案

完整的方案分为存储方案和跨域方案。如图 9-2 所示为 Storage 整体方案。

1. 存储方案：Store.js[5]——localStorage（标准浏览器 + IE 8+） + userData（IE 6/7）。
2. 跨域方案：使用 iframe 加载代理页，数据存储在代理页所在的域下，需要实现宿主页与代理页之间的通信，postMessage（标准浏览器 + IE 8+）+window.name（IE 6/7）。

[4] http://blogs.adobe.com/conversations/2011/11/flash-focus.html

[5] https://github.com/marcuswestin/store.js

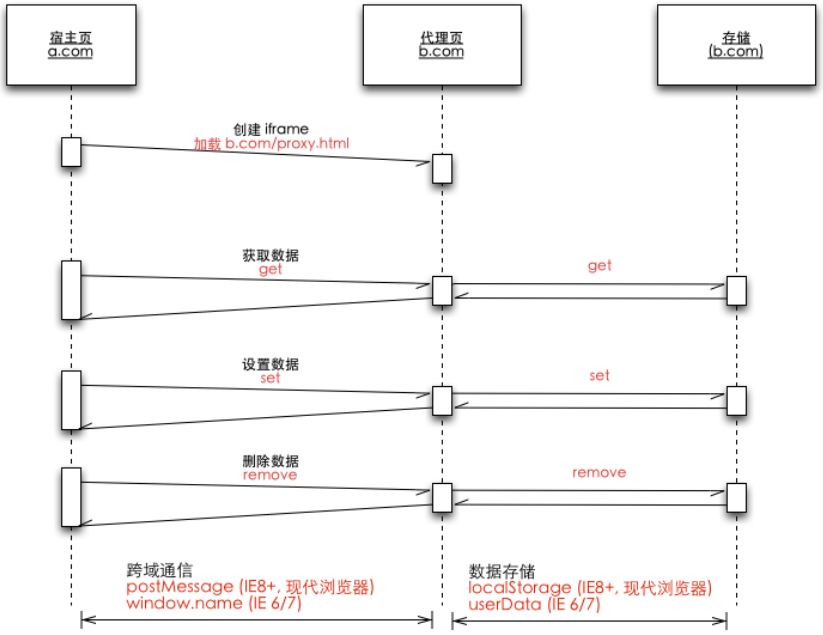



图 9-2　Storage 整体方案

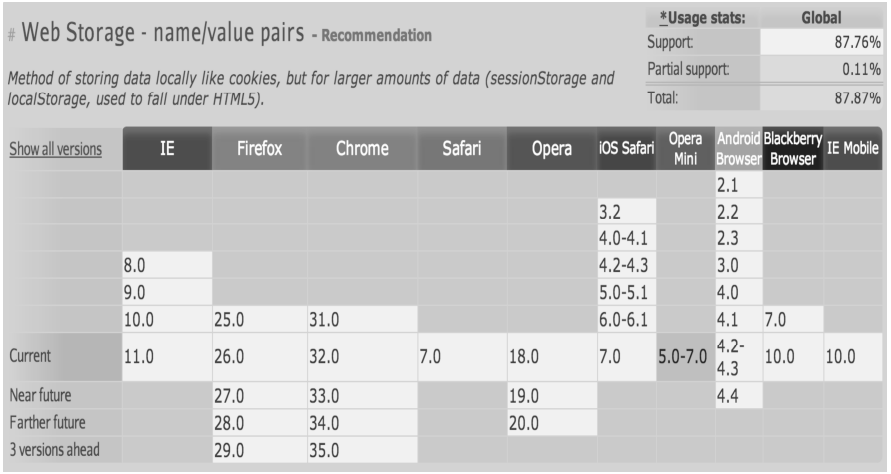
## 9.2.2　跨终端存储方案

**应考虑**

1. 首选 localStorage：
   a) 兼容性良好（见图 9-3），支持 IE 6/7 以外几乎所有 PC 和移动浏览器[6]。
   b) 有 Store.js 这样成熟的封装[7]。
2. userData 可以做 IE 6/7 的兼容方案。

[6] http://caniuse.com/#search=localstorage

[7] https://github.com/marcuswestin/store.js



# Web Storage - name/value pairs - Recommendation

*Method of storing data locally like cookies, but for larger amounts of data (sessionStorage and localStorage, used to fall under HTML5).*

| *Usage stats: | Global |
|---|---|
| Support: | 87.76% |
| Partial support: | 0.11% |
| Total: | 87.87% |

| Show all versions | IE | Firefox | Chrome | Safari | Opera | iOS Safari | Opera Mini | Android Browser | Blackberry Browser | IE Mobile |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 2.1 | | |
| | | | | | | 3.2 | | 2.2 | | |
| | | | | | | 4.0-4.1 | | 2.3 | | |
| | 8.0 | | | | | 4.2-4.3 | | 3.0 | | |
| | 9.0 | | | | | 5.0-5.1 | | 4.0 | | |
| | 10.0 | 25.0 | 31.0 | | | 6.0-6.1 | | 4.1 | 7.0 | |
| Current | 11.0 | 26.0 | 32.0 | 7.0 | 18.0 | 7.0 | 5.0-7.0 | 4.2-4.3 | 10.0 | 10.0 |
| Near future | | 27.0 | 33.0 | | 19.0 | | | 4.4 | | |
| Farther future | | 28.0 | 34.0 | | 20.0 | | | | | |
| 3 versions ahead | | 29.0 | 35.0 | | | | | | | |

图 9-3 localStorage 兼容性

**不考虑**

1. Flash 方案，Adobe 已经宣布放弃 Flash 移动端支持[8]，并且 iOS 中无 Flash 支持。
2. Cookie 存储上限只有 4KB，且会加重网络负担，Cookie 会被发送给作用域之内的所有 HTTP 请求，包括 JavaScript 请求、CSS 请求、图片请求，以及很多异步的请求等，如果按照 10 亿 pv 计算，1B Cookie 大约会增加 5.590GB 的流量，1KB Cookie 就会增加 5.590 TB 流量。Cookie 不适合作为存储载体。

**存储上限**

1. localStorage 5MB，有浏览器差异[9]。如图 9-4 所示是 localStorage 存储上限测试。
2. userData 单个文件 128KB，单个域 1024KB[10]。

[8] http://blogs.adobe.com/conversations/2011/11/flash-focus.html

[9] http://dev-test.nemikor.com/web-storage/support-test/_

[10] http://msdn.microsoft.com/en-us/library/ms531424（v=vs.85）.aspx_

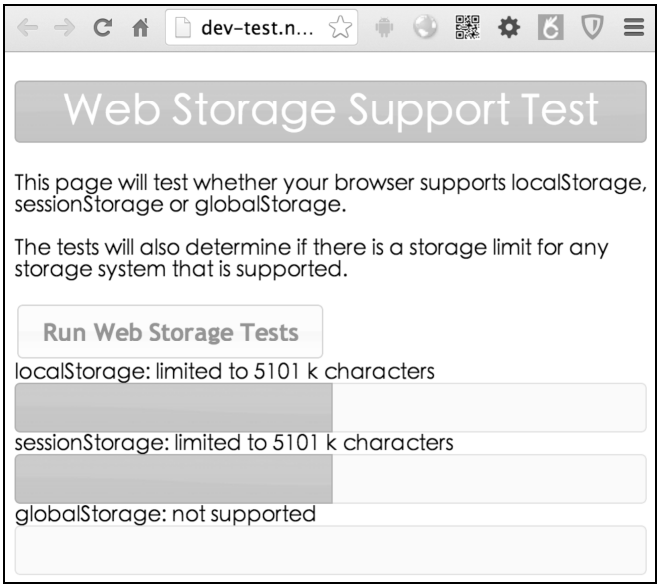



图 9-4　localStorage 存储上限测试[11]

## 9.2.3　跨域通信方案

跨域本质上还是通信问题，或说建立通信通道。表 9-1 罗列了最常见的 10 种[12]跨域通信方法，我们依据以下原则选择适合 Storage 的跨域通信方案。

1. iframe 场景下使用。
2. 不改写 document.domain。
3. 跨子域、跨主域。
4. 不使用 Flash。

[11] http://dev-test.nemikor.com/web-storage/support-test/

[12] 本节参考了 http://www.woiweb.net/10-cross-domain-methods.html 和 http://zciii.com/blogwp/crossdomain/。

表 9–1 Storage 跨域通信方案筛选

| | 通信形式 | iframe 场景 | 不改写 document.domain | 跨子域主域 | 不使用 Flash |
|---|---|---|---|---|---|
| JSONP | 单向通信 | ✘ | ✔ | ✔ | ✔ |
| window.name | 单向通信 | ✔ | （IE 6/7 例外） | ✔ | ✔ |
| CORS | 单向通信 | ✘ | ✔ | ✔ | ✔ |
| Flash URLLoader | 单向通信 | ✘ | ✔ | ✔ | ✘ |
| Server Proxy | 单向通信 | ✘ | ✔ | ✔ | ✔ |
| document.domain | 双向通信 | ✔ | ✘ | ✘ | ✔ |
| FIM | 双向通信 | ✔ | ✔ | ✔ | ✔ |
| Flash LocalConnection | 双向通信 | ✘ | ✔ | ✔ | ✘ |
| postMessage | 双向通信 | ✔ | ✔ | ✔ | ✔ |
| Cross Frame | 双向通信 | ✔ | ✔ | ✔ | ✔ |

满足所有条件的只有 postMessage、window.name（只用在 IE 6/7）。FIM 会改写浏览器地址有潜在的风险，Cross Frame 需要代理页部署在宿主页所在的域，实施成本过高也不考虑。

#### 9.2.3.1 异常处理

Storage 的所有调用都只有 success 一个回调，是基于这样的思路：“电梯永远不会坏”。电梯（扶梯）即使在断电或机械故障时并不影响人们的通行；Storage 也是如此，即使代理页或其他异常导致不能进行数据存取，那么 success 回调的参数为 undefined，并不会阻塞回调，更不会导致整个流程的阻塞。

#### 9.2.3.2 IE 6/7 兼容处理

尽管 IE 6/7 是要退出历史舞台的平台，为了能够让 Storage 兼容 IE 6/7 着实费了一番工夫。

IE 6/7 不支持 localStorage，需要用 userData 做降级存储方案。IE 6/7 不支持 postMessage，需要使用 window.name 进行跨域通信；window.name 方案其实就是代理页改写父页面的 window.name，父页面轮询 window.name；首先要解决的就是并发请求，我们利用队列机制，确保请求在高速并发条件下得到执行不会出现丢失。

### 9.2.4　安全性

由于数据存储在代理页（proxy）所在的域，数据可被所有能够加载代理页的宿主页面访问，所以保证安全关键在于服务器端控制代理页的访问来源。

如果不希望存储的数据被第三方访问，可以：

1. 代理页部署至特定服务器、服务器端控制访问来源（推荐白名单机制）。
2. 实例化 Storage 设置代理页地址参数 proxy。

### 9.2.5　遗留问题

比较遗憾的是，iOS 6.0+ 的 Safari 浏览器以及 OSX Safari 6.1+ 默认启用了“禁用第三方 Cookie”功能，localStorage 也受到该策略的牵连在浏览器重启后数据丢失。

此外，Storage 的代理页脚本目前较大，有优化的空间。Storage 是基于 Kissy[13]开发的，更广范围的传播应该考虑基于原生 JavaScript 或其他库，欢迎大家 fork 并改版，代码在笔者的 GitHub 上[14]。

## 9.3　使用

Storage 的使用非常简单：实例化、调用接口（get/set/remove/clear）。本节代码可从 Storage API 文档中获得，Storage 以单元测试的形式提供 Demo[15]。

---

[13] 阿里系的 JavaScript 库：http://docs.kissyui.com/。

[14] 代码地址：https://github.com/luics/storage。

[15] 单元测试即 Demo：http://gallery.kissyui.com/storage/1.1/demo/index.html。

### 9.3.1 实例化

实例化的代码如代码 9-1 所示。

**代码 9–1 实例化**

```
KISSY.use ('gallery/storage/1.1/index', function (S, Storage) {
      var storage = new Storage ({
         proxy: 'http://my-proxy-url'
      }) ;
}) ;
```

参数详解：

1. prefix key 前缀，详见下文“推荐命名”。
2. proxy 数据存储在代理页所在的域，选择不同的代理页可以让数据存储在不同的域下。
   - common：（默认）http://www.tmall.com/go/act/stp-tm.php。
   - tmall：http://www.tmall.com/go/act/stp-tm.php。
   - taobao：http://www.taobao.com/go/act/stp-tb.php。
   - {proxy-url}：proxy 页面部署在特定域名下的 URL，阿里系以外使用 Storage 时需要自行部署 proxy 页面。
3. onload 代理页加载成功的回调。

### 9.3.2 set/get

set/get 方法的代码如代码 9-2 所示。

**代码 9–2 set/get**

```
KISSY.use ('gallery/storage/1.1/index', function (S, Storage) {
    var storage = new Storage () ;
    storage.set ({k: 'key', v: 'value', success: function () {
        storage.get ({k: 'key', success: function (data) {
            console.log ('获取的值：', data) ;
        }}) ;
    }}) ;
}) ;
```

### 9.3.3　remove/clear

remove/clear 方法的代码如代码 9-3 所示。

**代码 9–3　remove/clear**

```
KISSY.use ('gallery/storage/1.1/index', function (S, Storage) {
    var storage = new Storage () ;
    // 删除数据'key'
    storage.remove ({k: 'key', success: function () {
    }}) ;

    // 清空所有字段
    storage.clear ({success: function () {
    }}) ;
}) ;
```

### 9.3.4　推荐命名

**命名规则：**

- 推荐小骆驼（首字母小写其他词首大写），如 myFieldName。

**防冲突机制：**

- 推荐类文件路径格式 xx/yy/zz。
- package name 可以直接作为前缀。
- 没有 package 的场景下可以使用应用/子应用/数据字段名。

# 10 动作同步

**这一章介绍了如何在多个终端间实现动作的同步。**

## 10.1 原理

### 10.1.1 案例

首先看一个跨终端演示的功能，如图 10-1 所示。


图 10-1 跨终端演示

在这个例子中，在手机端页面中切换 slide 的动作会同步到 PC 端的页面中。从而实现了至少以下几个新奇的功能。

1. 演讲者使用手机控制大屏幕上的 slide 切换。
2. 观众的 Mobile 设备或 PC 同样可以接入，查看同步的 slide。类似的例子还出现在在线协作平台中、在线互动游戏中。

## 10.1.2　动作同步

以上的例子都可以总结为下面的结构，如图 10-2 所示。

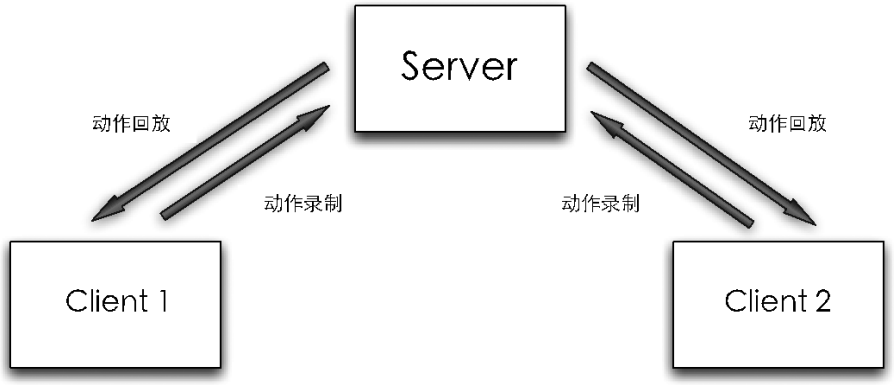



图 10-2　动作同步

图中涉及的对象：

- Server，服务器。
- Client 1，客户端 1，可以是移动设备或 PC 上的 Web 页面或 Native App。
- Client 2，客户端 2，定义同 Client 1。图中涉及的行为如下。
  - **动作**，在客户端上的任何操作，如 PC 上的鼠标点击、鼠标拖曳、鼠标移动，移动设备上的手指滑动、手指长按、手指轻敲等。
  - **动作录制**，将所有**动作**使用特定数据格式记录下来，**录制数据**通过长链接 C 传递到服务器 Server，再由 Server 传递给 Client。
  - **动作回放**，Client 接收到**录制数据**后，根据**数据格式 F** 解析数据后还原为**动作**。

## 10.2　实现

本节描述了脚本录制和回放的实现细节，尽管这些代码仍然是实验性质的，但是基于它读者可以构建更为强大的动作同步的应用。

本节所有代码可以从此处获取[1]，这是一个动作同步的完整 Demo。

1. 在 Mobile 浏览器中访问 http://{ip}:4000/mobile。
2. 在 PC 浏览器中访问 http://{ip}:4000/pc。
3. 在任何一个浏览器中点击色块，另一端的色块颜色同步变化。

### 10.2.1　Selenium

说起用户行为动作录制，我们就会想到 Selenium IDE，它的 Firefox 录制插件的强大和便捷性至今还让人印象深刻，Selenium 通过这种录制用户行为，然后在 Selenium Webdriver 中回放这个录制动作的方式，让 Web UI 相关的测试代码在很大程度上降低了复杂度、测试代码编写成本以及维护成本，让原本需要手写每一个交互行为的动作代码现在只需要按照交互行为顺序，录制一份动作行为，然后转换成约定的接口代码，并在回放录制脚本的过程中监控页面的错误以及做更加详细的校验，大致的过程如图 10-3 所示。

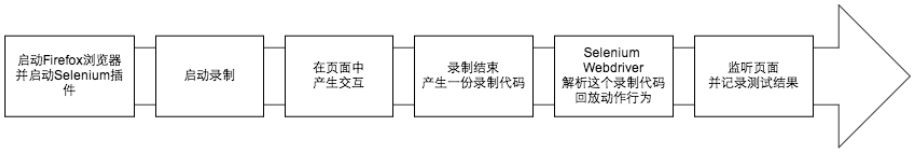



图 10-3　Selenium 工作流

Selenium 在这个环节中非常重要的一点就是，录制代码的接口约定，这个是类似于 HTTP 等的一种协议，根据这个约定的协议，工具录制出来符合协议的接口代码，然后回放工具按照这样的一份代码协议，完成协议的解析和动作的回放。

[1] http://luics.github.io/cew/actAync.zip

### 10.2.2　脚本录制和回放

在跨终端的应用中，也是类似于这样的一个过程。想象这样的一个场景，有两个客户端：桌面浏览器、Mobile 浏览器，两个端都访问同一个应用，那么如何做到在其中一个端做出了某个动作之后，另一个端也能同步地做出相应的操作。

上面的场景涉及在某个端的动作录制，然后在另外一个客户端回放，反过来亦然。只是现在动作的录制和回放都在纯浏览器中，并没有在 Selenium 中借助浏览器插件来录制动作。那么在纯浏览器中录制和回放动作，就只能通过 JavaScript 来完成。

在浏览器中，动作的产生是通过各种事件来完成的，事件在浏览器中有两种机制：冒泡和捕获。以冒泡为例，一个事件的产生默认总会往父元素传播，在这个传播的过程中，父级对象所有同类型的事件都会被触发，直到对象层级的顶层 document 对象，有些浏览器下是 window 对象。

基于以上对事件机制的分析，如果要录制页面动作，也就是监听各种事件的发生，根据冒泡机制，只需要在 document 或者 window 下绑定各种需要监听的事件即可。

在 PC 端的浏览器，一般比较常用的事件是 click、dbclick、mouseover、mouseout、mousemove 等，在 Selenium 里面就只有两种典型的情况：click 和 type。type 就是输入内容，click 就是发生点击事件的动作。在跨终端的情况下，PC 端和移动端的事件会存在或多或少的差异，一些在 PC 端常见的事件，到了移动端之后就不同了，移动端独有的 touch 类事件就类似于 PC 端的 click、mouse 类事件，当然还有独有的 gesture 类事件等。虽然看起来两个端浏览器的这些事件存在差异，但是如果通过一个中间事件类型映射转换，在动作的效果上是可以一致的。

回到上面所说的那个场景，PC 端浏览器和移动端浏览器访问同一个应用（当然也可以想象成是一个游戏），两个端的动作在操作的时候能保持一致。这里就涉及了在某个端录制动作行为，然后在另外一个端上回放的过程。为了能更加清晰地将该场景使用代码来说明，现在采用 Node.js+Express+Socket.io 来实现，并且以 click 和 touchstart 事件作为案例。

开头的时候说到 Selenium 在录制和回放所采用的一个接口“协议”，那么在这里也是有的，约定接口代码的形式，由于采用的是 Socket.io，利用它的事件机制来传递

数据，那么接口协议就是下面这种简单的 JSON 对象形式：

```
{ "device" : "pc|mobile", "action" : "click|touchstart", "target" :
"#example", "time" : "时间戳" }
```

device 就是设备类型；action 是事件，也就是动作类型；target 是目标元素的 css selector；time 是发生事件动作的时间。

有了上面的接口约定，那么接下来将要实现的大概步骤如图 10-4 所示。

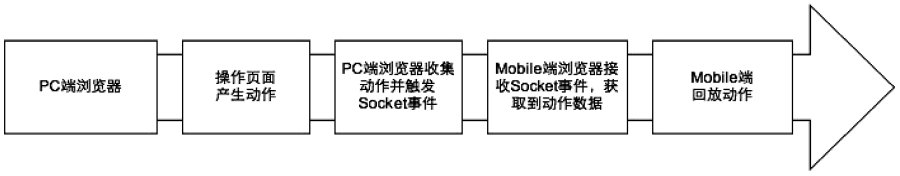



图 10-4　动作同步流程

首先，先来看看如何逐步实现 JavaScript 录制页面事件交互动作，只要在 document 上监听各种事件即可，这里以 click 事件为例，如代码 10-1 所示。

**代码 10–1　click 录制**

```
document.addEventListener ("click",function (e) {
    if (e.button === 0) {
        var target = e.targe;
        socket.emit ("send", { device : device, action : "click",
target : getSelector (target) , time : +new Date ()  }) ;
    }
},true) ;
```

在 document 监听到 click 事件后，通过 Socket.io 触发一个 send 的事件给服务端，然后由服务端再转发到其他端上去，执行相应的操作。

但是上面的代码会有一个很大的问题，这就是页面有很多无效的点击，也会冒泡到 document 中被监听到，这样就会导致 socket 所传输的很多事件数据是无效的，或者说并不是我们想要的，我们想要得到的是一个对于真正绑定了事件的元素触发事件后传输到各终端做相应的事件处理，那么这里就需要做一下事件监听的过滤。

我们需要判断 document 监听到的 click 事件的目标元素是否绑定了事件，处理方式如下面的代码 10-2 所示，先在页面 head 标签头部声明一段代码，重置 addEventListener 方法。

**代码 10–2　事件过滤**

```
var __addEventListener = Element.prototype.addEventListener;
Element.prototype.addEventListener = function (type,handler,capture) {
 if (!this['events']) { this['events'] = {}; }
 this['events'][type] = 1;
 return __addEventListener.apply (this,[type,handler,capture]) ;
}
```

上面代码的作用是，通过 addEventListener 函数绑定事件的 DOM 元素，会给它添加一个 events 的数组属性，这样我们就可以在 document 监听到 click 事件后，通过 target 元素判断它的 events 里面到底有没有 click 事件来过滤一些无效的 click 事件数据。

另外，我们还需要一个方法来判断该 target 元素是否有绑定事件，如代码 10-3 所示。

**代码 10–3　判断 target 绑定事件**

```
var findHasEventsElement = function (element,eventType) {
 if (!element.tagName) return null;
 if ( ( element['events'] && element['events'][eventType] ) ||
element.hasAttribute ("on" + eventType) || element["on" + eventType])
{
      return element;
 } else {
      if (element.parentNode != null) {
           return findHasEventsElement (element.parentNode,eventType) ;
      } else {
           return null;
      }
 }
}
```

findHasEventsElement 方法中会判断 3 种类型的事件绑定方式：通过 addEventListener、直接在 HTML 标签中使用 onxxx 声明，或者在 JavaScript 代码中直接给 onxxx 赋值来绑定事件。而且还会考虑另外的一种情况，就是采用了 delegate 的方式绑定的事件，寻找 target 父元素是否有绑定相应的事件。

通过这样的方式，可以很大程度上避免一些无效的事件数据的传输。

然后，对上面 document 所绑定的 click 事件监听函数做一下修改，如代码 10-4 所示。

**代码 10–4 click 录制**

```
document.addEventListener ("click",function (e) {
    if (e.button === 0) {
        var target = e.target,
            enable = findHasEventsElement (target,"click") ;
        if (enable) {
            socket.emit ( "send", { device : device, action :
"click", target : getSelector (target) , time : +new Date () }) ;
        }
    }
},true) ;
```

加上了 findHasEventsElement 函数的判断，只有检测到 target 元素绑定有事件，才会进行 socket 的事件数据传播。

同样的原理，对 touchstart 事件也是做相应的处理，如代码 10-5 所示。

**代码 10–5 完整的 touchstart 监听**

```
document.addEventListener ("touchstart",function (e) {
    var target = e.target,
        enable = findHasEventsElement (target,"touchstart") ,
        tagName;
    if (enable) {
        tagName = target.tagName.toLowerCase () ;
        socket.emit ( "send", { device : device, action :
"touchstart", target : getSelector (target) , time : +new Date () }) ;
    }
},true) ;
```

至此，我们对事件动作录制过程的代码实现已完毕，整合起来如下面的代码 10-6 所示。

**代码 10–6 click 和 touchstart 的完整录制**

```
var findHasEventsElement = function (element,eventType) {
 if (!element.tagName) return null;
 if ( ( element['events'] && element['events'][eventType] ) ||
element.hasAttribute ("on" + eventType) || element["on" + eventType])
{
    return element;
 } else {
    if (element.parentNode != null) {
        return findHasEventsElement (element.parentNode,eventType) ;
    } else {
```

```
            return null;
        }
    }
}

document.addEventListener ("click",function (e) {
    if (e.button === 0) {
        var target = e.target,
            enable = findHasEventsElement (target,"click") ;
        if (enable) {
            socket.emit ("send", { device : device, action : "click", target :
getSelector (target) , time : +new Date () }) ;
        }
    }
},true) ;

document.addEventListener ("touchstart",function (e) {
    var target = e.target,
        enable = findHasEventsElement (target,”touchstart”) ;
    if (enable) {
        socket.emit ("send", { device : device, action : "touchstart",
target : getSelector (target) , time : +new Date () }) ;
    }
},true) ;
```

上面的代码中还涉及一个 getSelector 的获取指定 element 的 CSS Selector，该方法实现如代码 10-7 所示。

**代码 10–7　getSelector**

```
window.getSelector = function (element) {
    var tagName = element.tagName.toLowerCase () ;
    function trim (string) {
        return string && string.replace (/^\s+|\s+$/,"") || string;
    }

    //去掉一些使用时间戳作为 ID 的元素
    if(element.id && !(/\d{13}/).test(element.id) && !(/^\d+$/).test
(element.id) ) {
        return '#' + element.id;
    }

    if (element == document || element == document.documentElement)
```

```
{
      return 'html';
  }

  if (element == document.body) { return 'html>' + element.tagName.
toLowerCase () ; }
  if(!element.parentNode){return element.tagName.toLowerCase();}
  var ix = 0,
      siblings = element.parentNode.childNodes,
      elementTagLength = 0,
      classname = trim (element.className) ;

  for (var i = 0,l = siblings.length; i < l; i++) {
      if (classname) {
          if (trim (siblings[i].className) === classname) {
              ++elementTagLength;
          }
      }else{
          if ( (siblings[i].nodeType == 1) && (siblings[i].tagName
=== element.tagName) ) {
              ++elementTagLength;
          }
      }
  }

  for (var i = 0,l = siblings.length; i < l; i++) {
      var sibling = siblings[i];
      if (sibling === element) {
         return arguments.callee ( element.parentNode ) + '>' +
 ( classname ? "." + classname.replace ( /\s+/g,"." ) :
element.tagName.toLowerCase () ) + ( (!ix && elementTagLength ===
1) ? '' : ':nth-child (' + (ix + 1) + ') ') ;
      }else if (sibling.nodeType == 1) {
         ix++;
      }
  }
};
```

接下来，我们还要准备两个页面，一个供 PC 端浏览器访问，另外一个供移动端浏览器访问，分别是 pc.html 和 mobile.html。把上面的代码加入到两个页面中，分别对两个页面的页面事件动作进行录制，两个页面的结构会比较类似，只是在事件绑定

上有所区别，pc.html 页面通过 click 绑定事件，mobile.html 通过 touchstart 绑定事件。

比如 pc.html 页面中，有一个 div 标签，id 为 example：

```
<div id="example"></div>
```

添加一些简单的样式：

```
#example{width:100px;height:100px;background:#f00;}
```

接着，给这个 div 标签绑定 click 事件：

```
document.getElementById ( "example" ) .onclick = function ( )
{ this.style.background = "#00c";
}
```

同理，mobile.html 页面中也会有 div 标签，只是绑定的是 ontouchstart 事件。还有一点，别忘记了把上面那段重置 addEventListener 函数的代码放到页面的 head 标签内。

现在，录制的代码逻辑以及页面都准备好了，但是还缺少一个重要的部分，这就是服务端 Socket.io 对两个端的事件数据传播。在 pc.html 中所传播的事件数据，broadcast 广播到 mobile.html 中，mobile.html 的事件数据广播到 pc.html 中，这样使得两个端的动作行为可以做出一致的响应。

比如 pc.html 中，初始化 socket.io 的逻辑，如代码 10-8 所示。

**代码 10–8　初始化 socket.io**

```
var socket = io.connect ('/') ;
socket.on ('connect', function (data) {
          //这里声明移动事件，监听来自 mobile.html 中广播的事件数据
  socket.on ("mobile", function (data) {
      //在这里处理来自移动端的事件
      var eventMap = {
          touchstart : "click"
      };

      $ (data.target) .trigger (eventMap[data.action]) ;
  }) ;
}) ;
```

同理在 mobile.html 中声明 PC 事件，监听来自 pc.html 的事件数据。然后把 click 事件映射为 touchstart 事件来触发。客户端已经准备好了，那么来看看服务端的处理，

其实就更加简单了，就是对接收到的数据中转一下，广播出去而已，如代码 10-9 所示。

**代码 10–9 socket.io 服务端初始化**

```
io.sockets.on ("connection", function (socket) {
  socket.emit ("connect", { msg : "success" }) ;
  socket.on ("send", function (data) {
      socket.broadcast.emit (data.device, data) ;
  }) ;
}) ;
```

服务端只需要绑定一个 send 方法，然后通过它中转数据，这样就使两个端的事件数据互相传输了。

在上面的例子中，动作回放部分比较简单，就是获取到相应的事件数据之后，trigger 触发一下相应的事件，因为上面所展示的例子实时性要求较高，所以在动作录制、传播、动作回放方面都是一步到位的，并没有像 Selenium 的那种，先通过工具录制好了动作数据代码，然后通过另外的回放程序来完成动作的回放。

如果对时效性要求不高，以及对数据大小传输上有极致的要求，那么上面的例子也是可以改造成先录制动作，等到缓存了一部分动作数据之后，再通过 socket 广播出去，然后再约定一个更加精简的接口协议，两个端来解析并回放动作。

上面所展示的支持简单的 click/touchstart 事件，其实 PC 和移动端的事件都是可以转化的，只是根据不同的设备特点，一个是鼠标，一个是多手势的屏幕操作而已。对于事件交互来说，实质上还是一样的。这方面应用的想象空间很大。

随着 HTML5 的普及以及浏览器对 HTML5 的支持更加丰富，跨终端的交互会变得越来越流行，比如当下广泛应用的 slide 演示，手机设备充当了 slide 的操作器，而展示却在 PC 端。或者这样的场景，一个游戏，可以跨终端玩，多个用户在一个游戏里面交互，而这些在纯浏览器端就可以完全实现。这些实现的基础环节，是浏览器对 Websocket 的支持，Websocket 是其中一个重要的桥梁，连接着多个终端，在未来的 Web 发展道路上，也会更加地广泛应用。

# A

# 附录 A　GBS

## 1. 浏览器测试基准和操作系统[1]

浏览器测试基准提供一套基准测试的浏览器。它旨在通过测试浏览器组合的最小可能子集，并利用共享的核心浏览器引擎的隐式覆盖率，以有限的测试资源最大化测试覆盖率。为了达到“基准”覆盖率，所有列出的浏览器至少应在一个操作系统中测试。一个特性如果含有特定操作系统下的兼容性问题且在单个操作系统下达到基准覆盖率时，才需要针对多个操作系统进行测试。测试操作系统的选择应该基于使用率和市场趋势。

浏览器测试基准定义了具有验证过的可用的用户体验的一组浏览器。试图为所有测试浏览器提供 A-级（见下文）体验既不经济也不常见。我们建议采用分层的办法对用户体验进行设计、开发和测试，并鼓励每个项目定义最适合于目标用户和测试资源的层级。

## 2. 分级浏览器支持：What&Why

回到 20 世纪 90 年代初，对于最初 10 年的专业 Web 开发而言，浏览器支持只有两种情况：支持或不支持。“支持特定的浏览器吗？”当时的回答显然是否定的，就连用户访问站点这种行为在当时也经常被阻止。随着 1998 年 IE 5 的发布，专业的 Web 设计师和开发者开始习惯于在给出任何决策之前问自己，“这个项目我必须支持 Netscape 4.x 吗？”

---

[1] Yahoo GBS（Graded Browser Support，分级浏览器支持），原文地址：http://yuilibrary.com/yui/docs/tutorials/gbs/。

相反，现代 Web 开发中必须支持所有的浏览器。选择排除部分用户是不合适的，但是有了“分级浏览器支持”策略后，就不再需要支持所有浏览器了。

分级浏览器支持有两个基本概念。

- 更广泛、更合理的“支持”的定义。
- 支持的“分级”概念。

### 3. “支持”的含义

支持并不意味着所有人看到完全相同的东西。希望两个不同的浏览器用户拥有相同的体验其实有悖于 Web 的异构本质。为所有用户提供完全相同的体验事实上会对参与性创造人为的障碍。内容的可用性和可访问行才是应该优先考虑的。

以电视机为例。电视机的核心价值是传播信息。即使手摇式应急收音机也能够接收电视的音频。即使是不完整的体验也比阻止接收音频要来得好。

一些观众使用黑白电视机。使用“最小公分母”方法，即只有黑白电视信号，虽然确保了共享的体验，但是对谁都无益。黑白电视以外的用户完全不能使用，好比“你得长这么高才能去驾驭”[2]的方法。

合适的支持策略应该允许每个用户尽可能多地利用环境提供的视觉和交互的丰富特性。这个方法（通常被称为渐进式增强）基于一个可访问的核心构建了丰富的体验。

### 4. 渐进式增强与优雅降级

优雅降级和渐进式增强是浏览器支持策略中常见的概念。它们是“容错”工程实践中两个密切相关的方法。

这两个概念影响了浏览器支持的决策。由于它们反映了不同的优先事项，在支持的讨论上它们也有着自己的框架。优雅降级允许较少使用的浏览器提供更少的内容给用户。渐进式增强以内容为核心，允许最多使用的浏览器展现更多内容给用户。虽然概念相近，渐进式增强却是更为健康和有前瞻性的方法。因而渐进式增强才是分级浏览器支持的核心概念。

---

[2] 原文是：“you must be this tall to ride”

## 5. 什么是支持的分级

虽然排除差异的浏览器支持是必要的，但会面临设计、开发、测试的挑战。一旦出现挑战，该如何判断体验是否突破了支持的集合？为了回答这个问题并且让系统更有秩序性，我们定义了支持的分级。有 3 个级别：A-级、C-级和 X-级。

在讨论每个级别之前，先看下对支持级别定义有帮助的一些特性。

可辨识的与未知的。目前有超过 10000 个浏览器品牌、版本和配置，并且这个数字还在不断增长。有必要对已知的浏览器进行分组。

有能力的与无能力的。没有哪两个浏览器拥有完全相同的实现。应该根据现代 Web 标准对浏览器进行分组。

现代与过时。随着新版本浏览器发行，较早版本在不断下降。

通用与稀少。虽然有成千上万的浏览器在使用，但只有少数几种被广泛使用。

## 6. 支持的 3 个级别

### 6.1　C-级

C-级是支持的基础级别，提供核心的内容和方法。有时被称作核心支持。只通过语义化的 HTML 定义，内容和体验高度可访问，可被装饰或高级功能削弱，具有很好的前向和后向兼容性。样式和行为层不在 C-级考虑范围。

C-级浏览器应该加入浏览器黑名单中。

总结：C-级浏览器是可辨识的，无能力的，陈旧而稀少的。QA 抽样测试 C-级浏览器，发现的 bug 应该设置为高优先级。

### 6.2　A-级

A-级是最高的支持级别。为了充分利用现代 Web 标准的强大能力，A-级体验包括了高级功能和视觉保真度。

A-级浏览器应放置于白名单中。Yahoo 96% 的用户可以享受 A-级体验。

总结：A-级浏览器是可辨识的，有能力的，现代而通用的。QA 测试所有 A-级浏览器发现的 bug 应该设置为高优先级。

### 6.3　X-级

X-级是为未知的、零散的、很少使用或停止开发的浏览器而准备的。X-级浏览器是有能力的。（如果一个浏览器是无能力的就应该被划分至 C-级，无能力的浏览器不具备现代浏览器的功能，它的用户体验在没有复杂样式和交互功能的情况下反而更好。）

未被归为其他级别的浏览器都是 X-级浏览器。

总结：X-级浏览器是有能力且现代的。QA 无须测试。

## 7. A-级和 X-级支持的关系

再说一点 A-级和 X-级浏览器之间的关系：一个 X-级的特殊例子是，A-级浏览器的一个新发布的版本。新品牌的（自然也是未测试的）浏览器未被定位为 A-级浏览器，但是显然 QA 的测试是一个 A-级需求。这个例子彰显了分级浏览器支持的力量。A-级和 X-级浏览器的实际差异在于 QA 会对 A-级浏览器进行测试。

不像 C-级只关注 HTML，X 级关注 A-级所关注的一切。尽管新品牌的浏览器可能会被划分为 X-级浏览器，但我们仍然给了它一个拥有和 A-级浏览器一样体验的机会。

## 8. 3 个级别之外

最近几年，我们发现在前述 3 个支持层级外又出现了大量新层级，这些新增的层级通常出现在某些浏览器实现的某些特性。定义和实现这些用户体验的层级应该由每个项目自身决定。总体来看，我们支持最简明的渐进式增强的方法，不鼓励不计开发、测试、维护代价地随意增加新的层级。

## 9. 质量保证（QA）测试

分级浏览器生态系统使得有意义的、目标明确的、经济的 QA 测试变得可能。如前所述，有选择的 C-级测试和系统的 A-级测试保证了绝大部分用户有可用的、得到验证的体验。A-级测试必须是细致而完整的，同时 C-级测试可以选择 1～2 个有代表性的浏览器（比如 Netscape 4.x 和 Lynx ）或者使用 CSS 和 JavaScript 禁用后的某个现代浏览器。

有必要重申：无须对 X-级浏览器进行测试。

针对核心体验有代表性的测试是很重要的。如果你的项目选择了分级浏览器支持，请首先确认站点的核心内容和功能能在不支持图片、CSS、JavaScript 的环境下被访问到。请确保键盘是完成任务所必需的，当 C-级浏览器访问时所有高级功能都将隐藏。

## 10. 结论

分级浏览器支持提供了一种支持的通用定义和一个驾驭持续增长的浏览和前端技术的框架。

万维网发明人和 W3C 的负责人 Tim Berners-Lee 认为它是最棒的：“任何人在页面上看到‘这个页面最适合 X-级浏览器下查看’都会想起过去的坏日子，那是在 Web 出现之前，你极少有机会查看在另一台计算机、另一个文字处理器或另一个网络上的文档”。

# B

# 附录 B　JSON Schema Core[1]

JSON Schema: core definitions and terminology

json-schema-core


## 摘要

基于 JSON 并用于定义 JSON 数据结构的 JSON Schema 定义了一种新的媒体类型“application/schema+json”。 JSON Schema 为特定应用程序提供了 JSON 数据的约定并规范了如何与之交互。JSON Schema 旨在为 JSON 数据定义校验、文档、超链接导航以及交互控制。


## 状态

这份互联网草案（Internet-Draft）的格式完全遵守 BCP 78 和 BCP 79。

互联网草案是 IETF 进行中的文档。其他工作小组也在分发互联网草案，所有当前的互联网草案列在此处 http://datatracker.ietf.org/drafts/current/。

互联网草案最长有效期 6 个月，可能在任何时间被其他文档更新、替换或作废。除非状态为“work in progress”，否则不推荐将互联网草案用作参考资料或引用它们。

本互联网草案将于 2013 年 8 月 3 日过期。

---

[1] http://tools.ietf.org/html/draft-zyp-json-schema-04

## 版权



## 1. 引言

JSON Schema 是用于定义 JSON 数据结构的 JSON 媒体类型。 JSON Schema 为特定应用程序提供 JSON 数据的约定并规范了如何与之交互。JSON Schema 旨在为 JSON 数据定义校验、文档、超链接导航以及交互控制。

本规范定义了 JSON Schema 核心术语和机制，相关的规范建立在此规范之上并定义了 JSON Schema 的不同应用。

## 2. 约定和术语

本文档中的关键字“MUST”、“MUST NOT”、“REQUIRED”、“SHALL”、“SHALL NOT”、“SHOULD”、“SHOULD NOT”、“RECOMMENDED”、“MAY”和“OPTIONAL”的详细描述见于 RFC2119。

“JSON”、“JSON text”、“JSON value”、“member”、“element”、“object”、“array”、“number”、“string”、“boolean”、“true”、“false”和“null”等术语的定义见于 RFC4627。

## 3. 核心术语

### 3.1. Property、item

当提及 JSON Object（定义在 RFC4627），术语“member”（成员）和“property”（属性）是可以互换使用的。

当提及 JSON Array（定义在 RFC4627），术语“element”（元素）和“item”（成员）是可以互换使用的。

### 3.2. JSON Schema、keywords

JSON Schema 是一个 JSON 文档，文档的内容必须（MUST）是一个对象。由 JSON Schema（本规范或相关规范）所定义的对象成员被称为关键字（keyword） 或 schema 关键字。JSON Schema 可以（MAY）包含 schema 关键字以外的属性。

### 3.3. 空 schema

一个没有任何属性或属性均为非 schema 关键字的 JSON Schema 被称为空（empty）schema。

### 3.4. 根 schema、子 schema

下面这个 JSON Schema 没有子 schema（subschema）：

```
{
    "title": "root"
}
```

JSON Schema 可以嵌套，如：

```
{
    "title": "root",
    "otherSchema": {
        "title": "nested",
        "anotherSchema": {
            "title": "alsoNested"
        }
    }
}
```

上面例子中的“nested”和“alsoNested”是子 schema，“root”是 root schema。

### 3.5. JSON Schema 原生类型

JSON Schema 为 JSON 值定义了 7 种原生类型（primitive type）。

- array：JSON array
- boolean：JSON boolean
- integer：JSON number，不包含小数和指数部分

- number：JSON number，该类型包括了“integer”
- null：JSON null
- object：JSON object
- string：JSON string

### 3.6. JSON 值判等

两个 JSON 值相等当且仅当：

- 均为 null；或
- 均为 boolean，且有相同的值；或
- 均为 string，且有相同的值；或
- 均为 number，且有相同的算术值；或
- 均为 array；且
  - 相同的数组成员数；且
  - 相同索引下的元素的值相等；或
- 均为 object，且
  - 属性名集合相同；且
  - 相同属性名下的元素的值相等

### 3.7. 实例

实例（Instance）可以是任何 JSON 值。一个实例可以由一个或多个 schema 描述，实例也被称为“JSON 实例”或“JSON 数据”。

## 4. 总览

本文档提出一种新的媒体类型“application/schema+json”，用于标识描述 JSON 数据的 JSON Schema。JSON Schema 自身也符合 JSON 格式。根据允许的值、文本描述以及它们与其他资源之间的关系，本规范和相关规范定义了一组关键字用于描述 JSON 数据。接下来的章节是对相关规范特性的一个总结。

### 4.1. 验证

JSON Schema 允许应用程序通过非交互和交互的形式验证实例。非交互形式，是指

应用程序可以直接校验所获得的数据是否满足特定的一组约束；交互形式，是指应用程序根据 JSON Schema 所描述的约束构建一个交互接口收集用户的输入[2]。

### 4.2. 超媒体和链接

JSON Schema 提供了一种方式提取实例到其他资源的链接关系，或者说将实例用超媒体形式表现。这种方式允许 JSON 数据转换为超媒体文本（如 HTML），并置于更大规模相关资源的上下文中[3]。

## 5. 整体考虑

### 5.1. 所有 JSON 值的适用性

实例可以是任何合法的 JSON 值（RFC4627）。既然如此，JSON Schema 不会强制要求实例必须是某个特殊的类型：JSON Schema 能够描述包括 null 在内的任何 JSON 值。

### 5.2. 编程语言无关

JSON Schema 是编程语言无关的。唯一的限制来自于 RFC4627 和宿主编程语言[4]本身。

### 5.3. JSON Schema 和 HTTP

本规范认为 HTTP（RFC2616）是互联网中占主导地位的协议，与 HTTP 相关的官方规范的价值同样是占主导地位的。本规范使用这些规范中的一个子集实现了一套机制，以便让 JSON 实例能够与一个或多个 schema 产生关联。

### 5.4. JSON Schema 和其他协议

JSON Schema 没有为 HTTP 之外的任何协议定义客户端—服务器的接口语义（semantic）。这些语义是应用相关的，或 JSON Schema 使用者们根据实际需要而达成的某些约定。

---

[2] 笔者注：如带校验的 Web 表单。

[3] 笔者注：部分协议对超文本的表述较晦涩，整段在表达“我能把 JSON 转为 HTML”。

[4] 笔者注：“宿主编程语言”指代一切实现 JSON Schema 功能或与之交互的语言，如 JSON Schema 工具所使用的语言。

### 5.5. 算术整数

一些编程语言以及它们的解析器使用了不同的内部机制来表示浮点数和整型数。因而出于可复用的考虑，无论是 JSON Schema 还是 JSON 实例中的 JSON 值都应该（SHOULD）确保算术整数如本规范所定义的形式去表示。

### 5.6. 扩展 JSON Schema

实现（implementation）可以（MAY）为 JSON Schema 定义额外的关键字。除了明确提供支持外，schema 作者不应该（SHALL NOT）期望这些额外的关键字得到每份实现的支持。实现应该（SHOULD）忽略那些不支持的关键字。

### 5.7. 安全考虑

schema 和实例均是 JSON 数据，因而所有的安全考虑都定义在 RFC4627。

## 6. 关键字“$schema”

### 6.1. 目的

“$schema”关键字既是 JSON Schema 的版本标识，又是描述特定版本 JSON Schema 的资源文件的 URI[5]。

这个关键字必须（MUST） 位于 JSON Schema 的根路径下，它的值必须（MUST）是 URI（RFC3986）和合法的 JSON Reference，这个 URI 必须（MUST）是绝对的和归一化的地址。这个 URI 所指向的资源必须（MUST）能够成功描述它自己。推荐（RECOMMEND）schema 作者在 schema 中添加该关键字。

以下为预定义的值：

- http://json-schema.org/schema# 当前版本的 JSON Schema 规范
- http://json-schema.org/hyper-schema# 当前版本的 JSON Hyper Schema 规范
- http://json-schema.org/draft-04/schema#
- http://json-schema.org/draft-04/hyper-schema#
- http://json-schema.org/draft-03/schema#

---

[5] 笔者注：JSON Schema 本身的格式也可以用 JSON Schema 自身来描述，比如 http://json-schema.org/draft-04/schema#。

- http://json-schema.org/draft-03/hyper-schema#

### 6.2. 自定义

使用自定义的关键字扩展 JSON Schema 时，schema 作者应该（SHOULD）为“$schema”定义一个 URI。这个自定义的 URI 不能（MUST NOT）是上面预定义的值。

## 7. URI 解析作用域和解引用[6]

### 7.1. 定义

JSON Schema 使用 JSON Reference[7]作为 schema 的寻址（addressing）机制。它从两方面扩展了本规范。

1. JSON Schema 提供了改写 Base URI 的机制，这个机制规定 Base URI 的改写必须依据“id”关键字。
2. 定义了一个专用的解引用机制扩展 JSON Reference，让其能够接受任意的片段部分。

在 schema 内部改变 URI 被称为定义了一个新的解析作用域。schema 的初始解析作用域就是 schema 自身的 URI；当 schema 不是从 URI 加载进来的，初始解析作用域就是空 URI。

### 7.2. 通过“id”关键字改变 URI 解析作用域

#### 7.2.1. 合法值

“id”的值必须（MUST） 是 string，同时必须（MUST）是合法 URI。且该 URI 必须（MUST）是归一化的，同时不能是（SHOULD NOT）空片段（#）或空 URI。

#### 7.2.2. 使用

“id”被用于改变解析作用域。当遇到 id 时，实现必须（MUST）立刻解析 id 并替换最近的一个父作用域。解析出来的 URI 将被用作这个子 schema 及其子 schema 的新解析作用域，直到遇到下一个 id 为止。

---

[6] 笔者注：解引用是指引用的逆操作，如“"$ref": "schema1"”中的“schema1”是一个引用，将“schema1”解析为 URI 并获取资源内容替换引用的过程就是解引用（dereferencing）

[7] http://tools.ietf.org/html/draft-pbryan-zyp-json-ref-03

当使用“id”改变解析作用域时，schema 作者应该（SHOULD）保证它在 schema 中的唯一性。

以下是这种 schema 的一个例子：

```
{
    "id": "http://x.y.z/rootschema.json#",
    "schema1": {
        "id": "#foo"
    },
    "schema2": {
        "id": "otherschema.json",
        "nested": {
            "id": "#bar"
        },
        "alsonested": {
            "id": "t/inner.json#a"
        }
    },
    "schema3": {
        "id": "some://where.else/completely#"
    }
}
```

使用 URI-encoded JSON Pointer[8]（开始于根 schema）表示的子 schema 定义了以下的解析作用域。

- #（文档根节点）
  - http://x.y.z/rootschema.json#
- #/schema1
  - http://x.y.z/rootschema.json#foo
- #/schema2
  - http://x.y.z/otherschema.json#
- #/schema2/nested
  - http://x.y.z/otherschema.json#bar

[8] http://tools.ietf.org/html/draft-ietf-appsawg-json-pointer-07

- #/schema2/alsonested
  - http://x.y.z/t/inner.json#a
- #/schema3
  - some://where.else/completely#

#### 7.2.3. 正则解引用和行内解引用

当需要通过解析作用域解析 URI 时，实现有两个操作模式可选。

1. 正则解引用
   - 实现解引用所有解析的 URI。
2. 行内解引用
   - 实现对当前 schema 内部的 URI 有选择地解引用。

实现必须（MUST）支持正则解引用，可以（MAY） 支持行内解引用。如考虑这个 schema：

```
{
    "id": "http://my.site/myschema#",
    "definitions": {
        "schema1": {
            "id": "schema1",
            "type": "integer"
        },
        "schema2", {
            "type": "array",
            "items": { "$ref": "schema1" }
        }
    }
}
```

当一个实现遇到“schema1”引用时，根据最近的父作用域解析出的 URI 为 http://my.site/schema1#。根据所选的解引用模式，这个 URI 的处理方式会有差异。

1. 如果使用正则解引用，实现会直接解引用该 URI 并获取 URI 的内容。
2. 如果使用行内解引用，实现将会意识到“http://my.site/schema1#”已经在当前 schema 中有定义了，然后选择使用合适的子 schema。

#### 7.2.4. 行内解引用与分段（fragment）

当使用行内解引用时，解析作用域可能会指向一个非空片段但又不是 JSON Pointer，比如：

```
{
    "id": "http://some.site/schema#",
    "not": { "$ref": "#inner" },
    "definitions": {
        "schema1": {
            "id": "#inner",
            "type": "boolean"
        }
    }
}
```

选择支持行内解引用的实现应该（SHOULD） 能够使用这种引用。实现可以选择使用正则解引用（来支持这种特殊情况），当然不是必须要支持的。

### 7.3. 安全考虑

行内解引用生产出的正则 URI 可能不同于 root schema 的正则 URI。schema 作者应该（SHOULD）保证使用正则解引用与行内解引用获得的内容是一致的。

分段不是 JSON Pointer 的那些扩展的 JSON Referencing，只有行内解引用才能处理。这类引用是用于向后兼容的，不可以（SHOULD NOT）在新的 schema 中使用。

## 8. HTTP 协议相关的推荐机制

大多数交互式的 JSON Schema 处理都是通过 HTTP 的。因而本节给出一些建议， 如何通过 HTTP 现有可用的机制实现实例与 schema 之间的关联。一个实例可以被一个或多个 schema 所描述。

### 8.1. 通过“Content-type”头关联

推荐（RECOMMEND）JSON 实例的响应头“Content-Type”中增加“profile”参数。profile 参数的值必须（MUST）是合法的 URI 并且该 URI 应该（SHOULD）指代了一个合法的 JSON Schema。MIME 类型必须（MUST）是“application/json”或其子类型。

下面是这种 header 的一个例子：

```
Content-Type: application/my-media-type+json; profile=http://example.
com/my-hyper-schema#
```

### 8.2. 通过“Link”头关联

当用“Link”头时，“rel”参数必须（MUST）使用“describedBy”（RFC5988 section 5.3）。“Link”头的目标 URI 必须指向一个合法的 JSON Schema。

下面是这种 header 的一个例子：

```
Link: <http://example.com/my-hyper-schema#>; rel="describedBy"
```

## 9. IANA 考虑

本规范提交的 JSON Schema MIME 媒体类型的定义如下。

- 类型名称：application。
- 子类型名称：schema+json。

## 10. 参考

### 10.1. 规范参考

[RFC2119] **Bradner, S.**, “**Key words for use in RFCs to Indicate Requirement Levels**,” BCP 14, RFC 2119, March 1997 （**TXT**, **HTML**, **XML**）.

### 10.2. 信息参考

[RFC2616] **Fielding, R.**, **Gettys, J.**, **Mogul, J.**, **Frystyk, H.**, **Masinter, L.**, **Leach, P.**, and **T. Berners-Lee**, “**Hypertext Transfer Protocol -- HTTP/1.1**,” RFC 2616, June 1999 （**TXT**, **PS**,**PDF**, **HTML**, **XML**）.

[RFC3986] **Berners-Lee, T.**, **Fielding, R.**, and **L. Masinter**, “**Uniform Resource Identifier （URI）: Generic Syntax**,” STD 66, RFC 3986, January 2005 （**TXT**, **HTML**, **XML**）.

[RFC4627] Crockford, D., “**The application/json Media Type for JavaScript Object Notation （JSON）**,” RFC 4627, July 2006 （**TXT**）.

[RFC5988] Nottingham, M., “**Web Linking**,” RFC 5988, October 2010 （**TXT**）.

[json-reference] Bryan, P. and K. Zyp, "**JSON Reference （work in progress）**," September 2012.

[json-pointer] Bryan, P. and K. Zyp, "**JSON Pointer （work in progress）**," September 2012.

[json-schema-03] Court, G. and K. Zyp, "**JSON Schema, draft 3**," September 2012.

## 变动日志

draft-00

- 初始草案。
- 从 JSON Schema draft v3 改进得到本规范。
- 使用了 JSON Reference、JSON Pointer。
- 定义“id”和 URI 分辨域。
- 新增了互操作考虑。

C

# 附录 C　JSON Schema Validation[1]

JSON Schema: interactive and non interactive validation

json-schema-validation

## 摘要

JSON Schema（application/schema+json）其中一个目标便是对实例（instance）验证。验证过程可以是交互式或非交互式的。如应用程序可以依据 JSON Schema 构建一个用户界面来生成交互式内容并在用户输入时校验[2]，或直接校验从不同数据源获得的数据。本规范描述了一组用于校验的 schema 关键字。

## 状态

这份互联网草案（Internet-Draft）的格式完全遵守 BCP 78 和 BCP 79。

互联网草案是 IETF 进行中的文档。其他工作小组也在分发互联网草案，所有当前的互联网草案列在此处 http://datatracker.ietf.org/drafts/current/。

互联网草案最长有效期 6 个月，可能在任何时间被其他文档更新、替换或作废。除非状态为“work in progress”，否则不推荐将互联网草案用作参考资料或引用它们。

---

1　http://tools.ietf.org/html/draft-fge-json-schema-validation-00

2　笔者注：比如一个带验证功能的 Web 表单。

本互联网草案将于 2013 年 8 月 3 日过期。

## 版权



## 1. 引言

JSON Schema 可要求 JSON 文档（实例）满足一组规则。这些规则具现为本规范描述的一组关键字。此外，本规范还描述了用于辅助交互实例生成的一组关键字。

本规范将使用 JSON Schema Core 中定义的术语。非常建议读者拥有一份 Core 文档的副本。

## 2. 约定和术语

本文档中的关键字 “MUST”、“MUST NOT”、“REQUIRED”、“SHALL”、“SHALL NOT”、“SHOULD”、“SHOULD NOT”、“RECOMMENDED”、“MAY”和“OPTIONAL”的详细描述见于 RFC2119。

本规范使用“容器实例”指代数组和对象实例。使用“子实例”指代数组元素和对象成员的值。

本规范使用术语“属性集”指代对象成员名称的集合，比如{ "a": 1, "b": 2 }的属性集为[ "a", "b" ]。

如果数组中找不到任何 2 个元素的值相等（参考 Core 规范），那么就称数组元素的值是唯一的（unique）。

## 3. 复用考虑

### 3.1. string 实例的校验

在 JSON string 中，“nul”字符（\x00）是合法的。无论潜在的编程语言环境是否有处理这个字符的能力，string 实例都有可能包含了该字符。

### 3.2. number 实例的校验

JSON 规范中没有定义乘积的边界值和数字的精度。JSON Schema 也没有定义类似的边界值。这意味着无论潜在编程语言是否能够处理，JSON Schema 处理的 number 实例可以是任意大的，且/或拥有任意大的小数部分。

### 3.3. 正则表达式

“pattern”和“patternProperties”使用了正则表达式表达约束。这些正则表达式应该（SHOULD）满足 ECMA262。

更进一步，考虑到正则表达式支持上的巨大差异，schema 作者应该（SHOULD）限制所使用的正则表达式标记（token）为以下值。

- JSON 规范（RFC4627）中定义的 Unicode 字符。
- 简单字符集（[abc]），范围字符集（[a-z]）。
- 字符补集（[^abc], [^a-z]）。
- 简单量词：“+”（1 个或多个）、“*”（0 个或多个）、“?”（0 个或 1 个）及其惰性版本（“+?”、“*?”、“??”）。
- 范围量词：“{x}”（出现 *x* 次）、“{x,y}”（出现至少 *x* 次，至多 *y* 次）、“{x, }”（出现至少 *x* 次）及其惰性版本。
- 输入起始符（“^”）和输入终止符（“$”）。
- 简单组合（“(…)”）与替换（“|”）。

最后，无论在开始还是结束部分，实现不能（MUST NOT）认为正则表达式是锚定的[3]。举例来说，“es”可以匹配“expression”。

[3] 笔者注：锚定即含有头部锚点（“^”）或尾部锚点（“$”）。

## 4. 通用验证考虑

### 4.1. 关键字和实例原生类型

一些验证关键字只能应用于一种或数种原生类型。当实例的原生类型不能被给定的关键字校验时，校验结果应该（SHOULD）是成功。

本规范根据一种或数种原生类型在不同章节来组合定义关键字。也有部分关键字是对所有实例类型都有效的。

### 4.2. 相互依赖的关键字

有时验证实例，一些关键字受其他关键字的存在与否影响。这类关键字会合并在一节内。

### 4.3. 关键字的默认值

一些关键字一旦缺失可以（MAY）被实现认为具有一个默认值。这类关键字的默认值会被提及。

### 4.4. 容器实例的验证

用于验证容器实例（数组和对象）的关键字不能验证子元素（数组成员和对象属性）。这些关键字中的一部分还含有计算子 schema 的信息。计算子 schema 的算法会在独立的章节展开。

值得注意的是数组成员只需要验证通过一个 schema，而对象成员可能需要不只一个 schema[4]。

## 5. 按实例类型排序的验证关键字

### 5.1. number 或 integer 类型的实例验证关键字

#### 5.1.1. multipleOf

a）合法值

“multipleOf”的值必须为 JSON number 且必须（MUST）严格大于 0。

b）验证通过的条件

实例值除以“multipleOf”的值为整数时，验证通过。

---

[4] 笔者注：参见“8. 子 schema 计算的参考算法”。

### 5.1.2. maximum 和 exclusiveMaximum

a）合法值

“maximum”的值必须（MUST）是 JSON number。“exclusiveMaximum”的值必须（MUST）是 JSON boolean。

如果存在“exclusiveMaximum”，“maximum”也必须（MUST）存在。

b）验证通过的条件

验证是否通过取决于是否存在“exclusiveMaximum”。

- 如果“exclusiveMaximum”不存在或值为 false，实例的值小于等于“maximum”的值时有效。
- 如果“exclusiveMaximum”值为 true，实例的值小于“maximum”的值时有效。

c）默认值

“exclusiveMaximum”未定义时则认为其值为 false。

### 5.1.3. minimum&exclusiveMinimum

a）合法值

“minimum”的值必须（MUST）是 JSON number。“exclusiveMinimum”的值必须（MUST）是 JSON boolean。

如果存在“exclusiveMinimum”，“minimum”也必须（MUST）存在。

b）验证通过的条件

验证是否通过取决于是否存在“exclusiveMinimum”。

- 如果“exclusiveMinimum”不存在或值为 false，实例的值大于等于“minimum”的值时有效。
- 如果“exclusiveMinimum”值为 true，实例的值大于“minimum”的值时有效。

c）默认值

“exclusiveMinimum”未定义时则认为其值为 false。

## 5.2. string 类型的验证关键字

### 5.2.1. maxLength

a）合法值

“maxLength”的值必须（MUST）是大于等于 0 的 integer。

b）验证通过的条件

当 stirng 长度小于等于“maxLength”的值时，验证通过。

RFC4627 定义 string 长度为字符的数量。

### 5.2.2. minLength

a）合法值

“minLength”的值必须（MUST）是大于等于 0 的 integer。

b）验证通过的条件

当 stirng 长度大于等于“minLength”的值时，验证通过。

RFC4627 定义 string 长度为字符的数量。

c）默认值

“minLength”未定义时则认为其值为 integer 0。

### 5.2.3. pattern

a）合法值

“pattern”的值必须（MUST）为 string 且为合法的 ECMA262 所定义的正则表达式。

b）验证通过的条件

正则表达式匹配了实例值，验证通过。重申：正则表达式不是隐式锚定的[5]。

## 5.3. array 类型的验证关键字

### 5.3.1. additionalItems 和 items

a）合法值

“additionalItems”的值必须（MUST）是 boolean 或 object 类型。如果是 object 则必须（MUST） 是一个合法的 JSON Schema。

---

[5] 笔者注：锚定即含有头部锚点（“^”）或尾部锚点（“$”）。

“items”必须是一个 obejct 或 array。如果是 object，则必须（MUST）为合法的 JSON Schema。如果是 array，所有成员必须（MUST）是 object，每个 object 必须是合法的 JSON Schema。

b）验证通过的条件

array 实例校验通过与否取决于这两个关键字，规则如下。

- 如果“items”不存在或者值为 object，无论“additionalItems”存在与否，校验总是通过。
- 如果“additionalItems”的值为 boolean true 或者为 obejct，校验总是通过的。
- 如果“additionalItems”的值为 boolean false 或者“items” 是 array，只有实例大小小于等于“items”的大小时，校验通过。

c）例子

下面的例子覆盖了“additionalItems”为 false“items”为 array 的情况，这是唯一一种可能出现验证失败的情况。

这是示例 Schema：

```
{
    "items": [ {}, {}, {} ],
    "additionalItems": false
}
```

对于这个 Schema 而言，下面的实例都是合法的：

```
[ ],
[ [ 1, 2, 3, 4 ], [ 5, 6, 7, 8 ] ],
[ 1, 2, 3 ];
```

而下面的实例都是非法的：

```
[ 1, 2, 3, 4 ],
[ null, { "a": "b" }, true, 31.000002020013 ]
```

d）默认值

这两个关键字未定义时均认为其值为空 schema。

#### 5.3.2. maxItems

a）合法值

“maxItems”的值必须（MUST）是大于等于 0 的 integer。

b）验证通过的条件

array 实例的大小小于等于“maxItems”的值时，验证通过。

#### 5.3.3. minItems

a）合法值

“minItems”的值必须（MUST）是大于等于 0 的 integer。

b）验证通过的条件

array 实例的大小大于等于“minItems”的值时，验证通过。

c）默认值

如果“minItems”缺失则认为其值为 0。

#### 5.3.4. uniqueItems

a）合法值

“uniqueItems”的值必须（MUST）是 boolean。

b）验证通过的条件

如果值为 false 则实例验证总是通过。如果值为 true，只有 array 所有元素均是唯一的（unique）时才算验证通过。

c）默认值

如果“uniqueItems”未定义时则认为其值为 false。

### 5.4. object 类型的验证关键字

#### 5.4.1. maxProperties

a）合法值

“maxProperties”的值必须（MUST）是大于等于 0 的 integer。

b）验证通过的条件

object 实例的属性个数小于等于“maxProperties”的值时，验证通过。

#### 5.4.2. minProperties

a）合法值

“minProperties”的值必须（MUST）是大于等于 0 的 integer。

b）验证通过的条件

object 实例的属性个数大于等于“minProperties”的值时，验证通过。

c）默认值

如果“minProperties”未定义时则认为其值为 0。

#### 5.4.3. required

a）合法值

“required”的值必须是含有至少一个元素的 array。每个元素的类型必须（MUST）是 string 且是唯一的。

b）验证通过的条件

object 实例的属性集包含“required”所有元素时，验证通过。

#### 5.4.4. additionalProperties、properties 和 patternProperties

a）合法值

“additionalProperties”的值必须（MUST）是 boolean 或 object。如果是 obejct 则为合法的 JSON Schema。

“properties”的值必须（MUST）是 object。每个成员的值必须（MUST）是代表 JSON Schema 的 object。

“patternProperties”的值必须（MUST）是 object。每个属性名称应该是一个合法的 ECMA262 正则表达式。每个属性值必须（MUST）是代表 JSON Schema 的 object。

b）验证通过的条件

object 实例验证是否通过取决于“additionalProperties”。

- 当值为 true 或 schema，验证通过。
- 当值为 false，验证算法见下文。

c）默认值

“properties”或“patternProperties”未定义时则认为其值为空 object。

“additionalProperties”未定义时则认为其值为空 schema。

d）当“additionalProperties”值为 false

在这个条件下，实例的验证取决于“properties”和“patternProperties”的属性集。本节中“patternProperties”的属性名称被简称为 regex。

第一步，收集以下集合。

```
s
待验证的实例属性集。
p
“properties”的属性集。
pp
“patternProperties”的属性集。
```

处理过程如下。

- 从集合 s 中移除所有在 p 中的元素。
- 从集合 s 中移除所有被集合 pp 的任一 regex 匹配的元素。当完成以上两步后 s 为空时，验证通过。

e）例子

下面这个 schema 就是个例子，

```
{
    "properties": {
        "p1": {}
    },
    "patternProperties": {
        "p": {},
        "[0-9]": {}
    }
}
```

这是用于验证的实例：

```
{
    "p1": true,
```

```
    "p2": null,
    "a32&o": "foobar",
    "": [],
    "fiddle": 42,
    "apple": "pie"
}
```

3 个属性集分别是：

```
s
  [ "p1", "p2", "a32&o", "", "fiddle", "apple" ]
p
  [ "p1" ]
pp
  [ "p", "[0-9]" ]
```

应用 2 步算法。

- 第 1 步后，“p1”从“s”中移除。
- 第 2 步后，“p2”（“p”匹配）、“a32&o”（“[0-9]”匹配）和“apple”（“p”匹配）从“s”中移除。

“s”集合仍有 2 个元素 “”和“fiddle”。因此验证失败。

#### 5.4.5. dependencies

a）合法值

“dependencies”的值必须（MUST）是 object。obejct 的每一个成员必须（MUST）是 object 或 array。

如果值为 object，则必须（MUST）为合法的 JSON Schema。这被称为 schema 依赖。

如果值为 array，则必须（MUST）有至少一个元素。每个元素必须（MUST）是 string 且必须（MUST）是唯一的。这被称为属性依赖。

b）验证通过的条件

**Schema 依赖**

对于所有 schema 依赖的组合（name 和 schema）来说，当实例有这样属性时，实例必须（MUST）通过 schema 的验证。值得一提的是这里是指实例本身通过验证，而不是指该属性的值。

**属性依赖**

对于每一个属性依赖的组合（name 和属性集）来说，当实例有这样的属性时，实例必须（MUST）具有和属性集相同的属性。

## 5.5. 任意类型的验证关键字

### 5.5.1. enum

a）合法值

enum 的值必须（MUST）是 array。这个 array 必须（MUST）含有至少一个元素，且所有元素必须（MUST）是唯一的。

array 的成员可以（MAY）是包括 null 在内的任何类型。

b）验证通过的条件

实例值等于 array 中的某个元素时，验证通过。

### 5.5.2. type

a）合法值

type 的值必须（MUST）是 string 或 array。如果是 array，则所有元素必须（MUST）是 string 且必须（MUST）是唯一的。如果是 string，则其值必须（MUST）是 Core 规范中定义的 7 种原生类型之一。

b）验证通过的条件

实例的原生类型是 type 的值定义的类型之一时，验证通过。重申：number 包括了 integer。

### 5.5.3. allOf

a）合法值

allOf 的值必须（MUST）是含有至少一个元素的 array。所有元素必须（MUST）是表示 JSON Schema 的 object。

b）验证通过的条件

实例通过 allOf 中所有 schema 的验证时，验证通过。

#### 5.5.4. anyOf

a）合法值

anyOf 的值必须（MUST）是含有至少一个元素的 array。所有元素必须（MUST）是表示 JSON Schema 的 object。

b）验证通过的条件

实例通过 anyOf 中至少一个 schema 的验证时，验证通过。

#### 5.5.5. oneOf

a）合法值

oneOf 的值必须（MUST）是含有至少一个元素的 array。所有元素必须（MUST）是表示 JSON Schema 的 object。

b）验证通过的条件

实例只通过 oneOf 中一个 schema 的验证时，验证通过。

#### 5.5.6. not

a） 合法值

not 必须（MUST）是表示 JSON Schema 的 object。

b）验证通过的条件

实例未通过 not 所描述的 schema 时，验证通过。

#### 5.5.7. definitions

a）合法值

definitions 值必须是 object。每个成员必须是合法的 JSON Schema。

b）验证通过的条件

definitions 不直接参与校验，但它提供了一个用于存放内联 JSON Schema 的标准场所。

比如下面这个 schema 在“definitions”的子 schema 中描绘了一组负整数。

```
{
    "type": "array",
    "items": { "$ref": "#/definitions/positiveInteger" },
    "definitions": {
```

```
        "positiveInteger": {
            "type": "integer",
            "minimum": 0,
            "exclusiveMinimum": true
        }
    }
}
```

## 6. 元关键字

### 6.1. title 和 description

#### 6.1.1. 合法值

两个关键字的值都必须（MUST）是 string。

#### 6.1.2. 目的

两个关键字都用于描述用户接口产生的数据。title 适合做短标题，而 description 可为 schema 的实例数据提供详细的解释。

两个关键字都可以（MAY）在根 schema 或任何子 schema 中使用。

### 6.2. default

#### 6.2.1. 合法值

对这个关键字的值没有任何约束。

#### 6.2.2. 目的

default 用于为关联的 schema 提供一个默认的 JSON 值。推荐（RECOMMEND）default 的值满足所关联的 schema。

default 可以（MAY）在根 schema 或任何子 schema 中使用。

## 7. “format”：语义化验证

### 7.1. 前言

单纯的结构化校验不能满足所有的校验需求。“format”关键字被用于可复用的语义化校验，这种校验是针对诸如 RFC 等外部规范所精确描述的一组值。

这个关键字的值被称为 format 属性。属性必须（MUST） 是 string。一个 format 属性通常只能校验一个特定的实例类型集合。如果实例值的类型不在这个集合中，校验应该（SHOULD）通过。

### 7.2. 实现条件

实现可以（MAY）支持“format”关键字。他们需要这么做：

- 应该（SHOULD）实现下面列出的 format 属性。
- 应该（SHOULD）提供一个禁用该关键字的选项。

实现可以(MAY)添加自定义 format 属性。双方约定除外，schema 作者不应该(SHALL NOT）期望其他实现支持 format 和（或）自定义属性。

### 7.3. 已定义的属性

#### 7.3.1. date-time

a）适用性

“date-time”适用于 string 实例。

b）验证

当 string 实例满足 RFC3339 section 5.6 的定义，验证通过。

#### 7.3.2. email

a）适用性

“email”适用于 string 实例。

b）验证

当 string 实例满足 RFC5322 section 3.4.1 的定义，验证通过。

#### 7.3.3. hostname

a）适用性

“hostname”适用于 string 实例。

b）验证

当 string 实例满足 RFC1034 section 3.1 的定义，验证通过。

#### 7.3.4. ipv4

a）适用性

“ipv4”适用于 string 实例。

b）验证

当 string 实例满足 RFC2673 section 3.2 所定义的“dotted-quad”ABNF 语法的 IPv4 地址，验证通过。

#### 7.3.5. ipv6

a）适用性

“ipv6”适用于 string 实例。

b）验证

当 string 实例满足 RFC2373 section 2.2 所定义的 IPv6 地址，验证通过。

#### 7.3.6. uri

a）适用性

“uri”适用于 string 实例。

b）验证

当 string 实例满足 RFC3986 所定义的 URI，验证通过。

## 8. 子 schema 计算的参考算法

### 8.1. 前言

用于子实例验证的单个 schema 或一组 schema 的计算方式受以下因素影响。

- 容器实例类型。
- 子实例在容器实例中定义的特性。
- 隐含在计算过程中的关键字的值。

此外，很重要的一点是计算过程中隐含的一个或多个关键字未定义时将被认为具有默认值，本节后续会重申这个观点。

### 8.2. 数组成员

#### 8.2.1. 定义特征

数组子实例的定义特征是其索引（index）。重申：数组索引从 0 开始。

#### 8.2.2. 隐含的关键字和默认值

这个计算中的两个隐含的关键字是：items 和 additionalItems。

其中任何一个未定义时，将被认为值为空 schema。此外，additionalItems 的值 true 被认为与空 schema 等价。

#### 8.2.3. 计算

a）当 items 为 schema

当 items 是 schema 时，无论索引和 additionalItems 为何值，子实例必须通过这个 schema 的验证。

b）当 items 为 array

这种情况下，schema 取决于索引。

- 当索引小于等于 items 的大小时，子实例必须通过 items 中对应 schema 的验证。
- 其他情况，必须通过 additionalItems 中定义的 schema 的验证。

### 8.3. 对象成员

#### 8.3.1. 定义特征

定义特征是其成员的属性名。

#### 8.3.2. 隐含的关键字

隐含在计算过程的 3 个关键字是：properties、patternProperties、additionalProperties。

如果 properties 和 patternProperties 未定义，则认为其值为空 object。

如果 additionalProperties 未定义，则认为其值为空 schema。此外值 true 和空 schema 等价。

#### 8.3.3. 计算

a）计算中的名称集合

计算使用如下的名称集合。

- m：成员属性名。
- p：properties 的属性集。
- pp：patternProperties 的属性集。其成员被简称为 regex。
- s：子实例的 schema 集。

b）第 1 步：properties 中的 schema

如集合 p 包含 m，则 properties 对应的 schema 添加到集合 s。

c）第 2 步：patternProperties 中的 schema

集合 pp 中的每个 regex 如果匹配了 m，则对应的 schema 添加至集合 s。

d）第 3 步：additionalProperties

当且仅当集合 s 为空时，additionalProperties 定义的 schema 添加至集合 s。

## 9. IANA 考虑

本规范没有 IANA 相关的影响。

## 10. 参考

### 10.1. 规范参考

[RFC2119] **Bradner, S.**, "**Key words for use in RFCs to Indicate Requirement Levels**," BCP 14, RFC 2119, March 1997 （**TXT**, **HTML**, **XML**）.

### 10.2. 信息参考

[RFC1034] Mockapetris, P., "**Domain names - concepts and facilities**," STD 13, RFC 1034, November 1987 （**TXT**）.

[RFC2373] **Hinden, R.** and **S. Deering**, "**IP Version 6 Addressing Architecture**," RFC 2373, July 1998 （**TXT**, **XML**）.

[RFC2673] **Crawford, M.**, "**Binary Labels in the Domain Name System**," RFC 2673, August 1999 （**TXT**）.

[RFC3339] **Klyne, G., Ed.** and **C. Newman**, "**Date and Time on the Internet: Timestamps**," RFC 3339, July 2002 （**TXT**, **HTML**, **XML**）.

[RFC3986] **Berners-Lee, T.**, **Fielding, R.**, and **L. Masinter**, "**Uniform Resource Identifier （URI）: Generic Syntax**," STD 66, RFC 3986, January 2005 （**TXT**, **HTML**, **XML**）.

[RFC4627] Crockford, D., “**The application/json Media Type for JavaScript Object Notation （JSON）**,” RFC 4627, July 2006 （**TXT**）.

[RFC5322] **Resnick, P., Ed.**, “**Internet Message Format**,” RFC 5322, October 2008 （**TXT**, **HTML**, **XML**）.

[ecma262] “**ECMA 262 specification**.”

## 变动日志

draft-00

- 初始草案。
- 从 JSON Schema draft v3 改进得到本规范。
- 重新定义“required”关键字。
- 移除关键字“extends”、“disallow”。
- 新增关键字“anyOf”、“allOf”、“oneOf”、“not”、“definitions”、“minProperties”、“maxProperties”。
- “dependencies”成员值不再是一个字符串，依赖数组至少有一个成员。
- “divisibleBy”重命名为“multipleOf”。
- “type”数组不再包含 schema，“any”不再是合法取值。
- 重新整理了“format”一节，支持要求改为可选。
- “format”：移除属性“phone”、“style”、“color”，“ip-address”重命名为“ipv4”，为所有属性添加了参考。
- 为 array/object 实例提供计算 schema 的算法。
- 添加互操作考虑。

# 附录 D　if-spec 2.0

该附录是描述 if-spec 2.0 的 JSON Schema。

```
{
"$schema": "http://json-schema.org/draft-04/schema#",
"type": "object",
"description": "数据接口文档",
"required": ["meta", "request", "response"],
"properties": {
    "request": {
        "type": "object",
        "description": "请求格式，JSON Schema 格式"
    },
    "response": {
        "type": "object",
        "description": "默认响应格式，JSON Schema 格式"
    },
    "meta": {
        "type": "object",
        "description": "配置定义",
        "required": ["name"],
        "properties": {
            "name": {
                "type": "string",
                "description": "接口名称"
            },
            "description": {
                "type": "string",
                "description": "接口描述"
            },
```

```
            "type": {
                "type": "array",
                "description": "接口支持的类型集合，如 HTTP Method",
                "minItems": 1,
                "uniqueItems": true
            },
            "uri": {
                "type": "string",
                "format": "uri",
                "description": "线上接口地址"
            },
            "version": {
                "type": "string",
                "description": "服务器最低版本"
            },
            "responseMap": {
                "type": "array",
                "description": "响应格式匹配规则",
                "items": {
                    "type": "object",
                    "required": ["rule", "schema"],
                    "properties": {
                        "rule": {
                            "type": "array",
                            "description": "匹配规则",
                            "minLength": 1,
                            "items": {
                                "type": "object",
                                "required": ["type"],
                                "properties": {
                                    "type": {
                                        "type": "string",
                                        "description": "规则类型",
                                        "enum": ["method", "request",
"response"],
                                        "uniqueItems": true
                                    },
                                    "method": {
                                        "type": "string",
                                        "description": "type 为 method 时,
指定 HTTP Method，为 RESTful 预留",
                                        "enum": ["GET", "POST", "PUT",
```

```
"DELETE"]
                                    },
                                    "property": {
                                        "type": "string",
                                        "description": "type 为 request
时指定 URL 参数的名称；type 为 response 时指定成员名称；"
                                    },
                                    "data": {
                                        "type": "string",
                                        "description": "type 为 request
时，指定 POST data 成员"
                                    },
                                    "value": {
                                        "description": "不同场景含义不同"
                                    },
                                    "pattern": {
                                        "type": "string",
                                        "description": "正则表达式版本的
value"
                                    }
                                }
                            }
                        },
                        "schema": {
                            "type": "string",
                            "description": "匹配的响应数据格式"
                        }
                    }
                },
                "uniqueItems": true
            }
        }
    }
}
}
```

# 作者简介

鬼道（原名徐凯），2011 年毕业于同济大学计算机科学与技术系，模式识别方向硕士研究生。

现就职于天猫，先后负责天猫跨终端 Web 的业务推进和技术基础设施建设、天猫前端会员营销组。目前关注的领域是 Mobile Web、PC Web 和 Native App 的融合。

- 2014 年 3 月，获天猫技术部 2013 年度“最佳新人奖”，同时获“创新奖”提名。
- 2014 年 3 月，推动了天猫前端 16 项专利申请，其中 6 项通过集团法务审批，2 项专利局已受理。其中鬼道有 4 项专利通过集团法务审批，1 项专利局已受理。
- 2014 年 2 月，“IF——端到端的接口规范”入选“2013 阿里技术那些事”前端交互与设计领域的十大事件。
- 2013 年 11 月，W3CTECH 2013 主题分享“移动优先的跨终端 Web”。
- 2013 年 10 月，凭借跨终端跨域存储组件 Storage 获得“2013 Kissy Gallery 组件大赛”冠军。
- 2013 年 7 月，D2 主题分享“移动优先的跨终端 Web”。

鬼道曾就职于百度移动变现团队，所带领的前端团队负责 Web 和 Native App 的工作，此团队先后发布了：移动广告 Native SDK 和 Web SDK、移动富媒体广告 SDK、移动统计、移动网盟推广、移动网盟等多个产品。作为移动变现团队的中坚力量，获得了 2011 年度“百度最佳新人”。